Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2018

Marco Filipe Pinheiro Rosa

Segurança de Informação para Serviços de Apoio Social

Information Security for Social Support Services

Universidade de Aveiro
2018

Departamento de Eletrónica,
Telecomunicações e Informática

Marco Filipe Pinheiro Rosa

Segurança de Informação para Serviços de Apoio Social

Information Security for Social Support Services

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor João Paulo Barraca, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor André Zúquete, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático, Universidade de Aveiro

vogais / examiners committee

Professor Doutor António Alberto dos Santos Pinto
Professor Adjunto, Instituto Politécnico do Porto

Professor Doutor João Paulo Silva Barraca
Professor Auxiliar, Universidade de Aveiro

agradecimentos /
acknowledgements

O trabalho apresentado na presente dissertação foi parcialmente suportado pelo COMPETE - Programa Operacional Competitividade e Internacionalização (COMPETE 2020), Sistema de Incentivos à Investigação e Desenvolvimento Tecnológico (SI I&DT), no âmbito do projeto Social Cooperation for Integrated Assisted Living (SOCIAL).

Agradeço toda a ajuda de todos os meus colegas, companheiros e professores durante este percurso, com destaque ao meu orientador, o professor João Paulo Barraca, que sempre se mostrou disponível para me ajudar e incentivou-me a dar o meu melhor durante o desenvolvimento desta dissertação.

Palavras Chave

segurança, blockchain, integridade, logging, auditoria, controlo de acesso, autorização, ABAC, XACML, FHIR, HL7, registos de saúde electrónicos, cuidados de saúde.

Resumo

Cada vez mais, as pessoas querem a sua informação disponível digitalmente, visto que é mais cómodo, económico e rápido aceder do que o formato físico. No entanto, podem existir consequências ao adoptar esta medida. Como prevenimos que certas pessoas não consigam aceder a dados mais sensíveis ou como conseguimos garantir que não houve manipulações não autorizadas sobre certa informação? Ao desenvolver um programa que lida com transmissões de dados privados, estas são umas das várias preocupações de qualquer programador e devem ser bem pensadas e resolvidas. Neste documento iremos analisar uma solução possível para um dos mais importantes componentes de qualquer plataforma, o logging/auditoria e respectivo controlo de acesso, que apresenta uma tecnologia que ainda não se encontra muito presente fora da sua principal área de aplicação. O objetivo é que se dê a entender que é possível usar como base uma tecnologia que foi criada para um propósito específico e aplicá-la em cenários totalmente diferentes. A plataforma que será usada como base para aplicar estas ideias e conceitos será uma aplicação cujo propósito é prestar auxílio aos seus utilizadores e permitir a comunicação rápida e direta entre estes e os seus respetivos cuidadores e que também lidará com registos de saúde electrónicos.

Keywords security, blockchain, integrity, logging, auditing, access control, authorization, ABAC, XACML, FHIR, HL7, healthcare, electronic health records.

Abstract

Over the years, more people want their information available digitally, since it is more convenient, economical and faster to access than the physical format. However, there may be consequences in taking this action. How do we prevent certain people from gaining access to more sensitive data, or how do we ensure that there is no unauthorized manipulation of certain information? When developing a program that deals with private data transmissions, these are one of the many concerns of any programmer and should be well thought out and resolved. In this document we will analyze one possible solution for one of the most important components of any platform, the logging/auditing and respective access control, where one presents a technology that is not yet widely present outside its main area of application. The goal is to make it clear that it is feasible to use a technology that has been created for a specific purpose and to apply it in totally different scenarios. The platform that will be used as a basis for applying these ideas and concepts will be an application whose purpose is to assist its users and allowing them to communicate quickly and directly between them and their respective caregivers and that also will handle electronic health records.

Contents

List of Figures

List of Tables

Glossary

XACML	XML Access Control Markup Language
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
ABAC	Attribute-based Access Control
RBAC	Role-based Access Control
XML	eXtensible Markup Language
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PAP	Policy Administration Point
PRP	Policy Retrieval Point
HL7	Health Level Seven International
EHR	Electronic Health Records
FHIR	Fast Healthcare Interoperability Resources
DBMS	Database Management System
P2P	Peer-to-Peer
IDS	Intrusion Detection Systems
2LBC	2-layer BlockChain
SAML	Security Assertion Markup Language
SOAP	Simple Object Access Protocol
SP	Service Provider
IdP	Identity Provider
OASIS	Organization for the Advancement of Structured Information Standards
JWT	JSON Web Tokens
SOCIAL	Social Cooperation for Integrated Assisted Living
URI	Uniform Resource Identifier
API	Application Programming Interface
REST	Representational State Transfer
VM	Virtual Machine
IoT	Internet of Things
QoE	Quality of Experience
IP	Internet Protocol
L2TAP	Linked Data Log to Transparency, Accountability and Privacy
WAR	Web Application Archive
SOAP	Simple Object Access Protocol
TLS	Transport Layer Security
SSL	Secure Sockets Layer

CHAPTER 1

Introduction

1.1 Context

Over the years, the quality of life of several people has been greatly improved and facilitated through the integration of technology into several everyday products that help their users with their day-to-day activities. But, unfortunately, as technology is evolving so are the attackers' methods and it becomes more difficult to prevent those breaches. Sometimes their attacks can be completely hidden, and they can continue with their illegal activities for long periods of time. So, it is important to create solutions that can avoid these kind of issues, and avoid important and private data to be manipulated, or fall into the wrong hands.

The security methods that are going to be discussed here have in mind a platform that handles the exchange of healthcare information, but they can also be applied to any kind of service or industry, since interoperability is something that is valued when talking about security procedures.

1.2 Objectives

Information security is going to be the thesis main concern, but it will be more focused on the storage of logs (the events that occur inside a system) and the respective auditing (to provide the possibility to retrieve and analyze those events, so that the behavior of a user or service can be observed, and also to detect possible problems) that are integrated with the project (Social Cooperation for Integrated Assisted Living (SOCIAL)) that is associated with this dissertation. So, the main objective will be to create a logging and auditing service that can work independently from the other services that will exchange information with it (internal or external to the platform).

This main objective will have two goals. One is to guarantee the integrity of the stream of the log events that are stored on this service and to be able to easily detect attempts of

tampering that want to hide certain activities. The other is to create an access control for the log events to deny unauthorized actions (like writing or reading) over them from someone or something that does not have that kind of privilege.

With this document, the reader can have some context and knowledge over the already existing security procedures and why the solutions that are going to be implemented here are one of the better alternatives to solve some issues that appear with the more used methods.

1.3 Contributions

The thesis main purpose is (after analyzing it) to help better understand how important is to properly develop a logging and auditing service with an access control unit included, by providing several methods and examples of all of the different components that should be implemented into it.

The solution and implementation discussed through chapter 3 and 4 were developed to integrate a project (SOCIAL) whose main goal is to improve the quality of assistance for social services. When it is fully developed, the logging and auditing service will help the SOCIAL system administrators, auditors and even health experts to review the flow of activities inside the system and easily detect eventual problems and supervise certain users behaviors, that will help them improve over the assistance that they give them. The proposed solution of this document also guarantees that this information is consistent and that can only be accessed and audited by those who have all the authorization credentials to do so. Even though the solution proposed had in mind the uses cases from that particular project, it can be easily adapted into other types of projects, since we try to make it as generic as possible, for interoperability purposes.

1.4 Document's Structure

This document is divided into six chapters, so that the readers can have a better understanding of the different issues and themes related to the dissertation and to provide a quick and direct way of searching for specific topics.

The current chapter (1) introduces the topics that will be discussed over the whole document and provides an overview of what are the goals and objectives that are trying to be accomplished. Chapter 2 presents existent technologies and solutions that try to avoid security breaches, and that help maintain a system's stability, and how sometimes they are not the best approach to follow, for different types of systems. On Chapter 3 the solution that was used to solve many of the issues presented on 2 is going to be explained (including its architecture, requirements and use cases). The implementation of that solution is described on Chapter 4 and shows how the goals from the proposed solution can be attained and what existent technologies can be integrated with it. The scenarios and tests that will prove if the solution is viable or not are detailed on Chapter 5. Chapter 6 will conclude the document and point out possible improvements over the solution that was discussed over the previous chapters.

Additionally, there are some examples of the consequences for weak security implementations, to clarify how crucial it is to develop a robust and secure system for gaining trust and more clients to someone's services, available at this document's appendix.

CHAPTER 2

Context and Related Solutions

When developing a software that follows the client-server model, the work involved should be focused on two sides, the presentation and the data access (front-end and back-end, respectively). While the front-end targets what the client will see, the back-end aims its attention to the server-side of the product and how it will work.
On this paper, the focal point will be the back-end, primarily the security features of that layer.

This chapter introduces and explains several security concepts that are currently used and implemented on nowadays systems and also what are the best choices of all of the existent alternatives, depending on the goals that are trying to be accomplished.

The first sections, 2.1, 2.2 and 2.3, introduce more broad approaches to deal with specific security concerns (that are going to be implemented with the document's project), respectively how health records are exchanged on-line, how different services that communicate among themselves can authenticate and prove that they are truly who they claim to be and how data can maintain its integrity. The following sections, 2.4, 2.5 and 2.6, are topics more focused on the solution that will be proposed in the following chapter and discuss what is and how the logging and auditing processes can be used to benefit a system and how it is possible to deny unauthorized accesses to certain digital information.

2.1 Management of Electronic Health Records

Health related data needs to be exchanged as fast as possible and needs to be up to date, so that medical experts can get that information to fulfill their work's goals. However, physical medical records start to become inconvenient, since having these documents on someone hands can take up too much time and can even be outdated. With Electronic Health Records (EHR), it is possible to quickly retrieve and store patient's data, since that information is now in a digital format. This revolutionized the healthcare industry, because experts can now

access a vast range of data (demographic, medical, radiology images, ...) in a efficient way and also modify them in a quick manner for other people to see the most recent updated results.

Many researchers and developers found innovative ways to utilize EHR to improve the Quality of Life of patients. Yang et al. [125] explain how EHR can help discover certain patterns of diseases, and help find significant influential factors for a specific medical condition, by using its values into Poisson models to create a model of that data to analyze it against different characteristics. Of course, to totally take advantage of this data, the integrity must be assured. Vimalachandran et al. [122] warn that incomplete or inaccurate information inside EHR, that can be caused by attacks on the system, can lead to the death or worsen the condition of patients. The authors propose using the pseudonymisation technique on certain EHR values (see example of Figure 2.1), so that the actual data is replaced by pseudonyms, that are also stored in independent tables with the respective correct information, to reduce the risk of exposure. With this technique, the information cannot be accessed directly and changed without the proper authorization and the data remains private, since the pseudonym's true values are held elsewhere.

Healthcare Identifier	Medication	Date	Condition	Name
8001567898761234	Insulin	01-10-2014	CD	John Smith
8008123456785000	Dapotum	05-10-2014	MH	Jane Doe
8001567898761234	Thalitone	10-10-2014	CKD	John Smith

Pseudonymisation

Healthcare Identifier ID	Medication	Date	Condition	Name
0102	Insulin	01-10-2014	CD	A12
452	Dapotum	05-10-2014	MH	B02
2712	Thalitone	10-10-2014	CKD	N17

Figure 2.1: Basic pseudonymisation technique [122].

Since many services deal with health information, a secure way to exchange those health records is necessary. However most systems that manage EHR suffer from interoperability problems and whenever there is a need to exchange data between different systems, incompatibility problems appear, whether that be from the data type format, different structures of data and lack of relevant fields or protocols to follow. To help with that, in 2011, the Health Level Seven International (HL7) [109] introduced the first draft of the Fast Healthcare Interoperability Resources (FHIR) [48] standard and in 2014 launched it. HL7 is a health-care

standards organization which main goal is to provide a framework for the management of EHR and to allow the communication and exchange of those records between different systems.

The FHIR standard manages to solve the interoperability problem mentioned above, that many service developers encountered while dealing with EHR since it can manage eXtensible Markup Language (XML), JavaScript Object Notation (JSON), Hypertext Transfer Protocol (HTTP), Atom and OAuth structures and it can also be implemented into a service-based architecture, thanks to its' HTTP-based Representational State Transfer (REST)ful protocol [52].

Some examples of scenarios, when using FHIR (according to the official documentation [56]):

- When someone searches for patients, without any type of criteria, they get (as the result) all of their accessible patients.
- It is possible, using existing data, to generate an alarm message concerning a patient state.
- Requesting decisions, for example, "what immunizations should this patient have?". The response is a resource (whether the decision support engine can provide the requested decision or not).

Many organizations all over the world are showing interest in using FHIR and some of them are listed in the HL7's wiki page [54]. Some of these organizations are the Ministry of Health and Long Term Care (Ontario) [85], the Vitoria City Hall (Brazil) [16], the National E-Health Transition Authority (Australia) [9], the NHS Digital (England) [36] and the Orion Health (New Zealand) [70].

The FHIR data is represented as *Resources* [49], JSON objects with several fields and values relevant to the HealthCare industry, like *Patient*, *Practitioner*, *DiagnosticReport*, *Coverage*, *Questionnaire*, *Consent* and so on. They can have six different classifications: *Clinical*, *Identification*, *Workflow*, *Financial*, *Conformance* and *Infrastructure* and can be aggregated (through certain fields and values) into blocks, called *Bundles*.

Although all of these resources must follow a certain structure with specific fields and type of data that must be included, there is a type of resource that does not need to follow specifications established by HL7 standards, the *Basic* resource. It can be created or used in different circumstances, for example, for resources that do not yet have a structure established by HL7 or when it is necessary to describe a resource that does not exactly follow the rules established by the original structure (defined by HL7) of that type of resource.

Certain resources may also have *Security Labels* [51] that can have three different purposes: to show the permissions for operations like read, modify and others, to show what resources can be returned and to show how the data should be dealt with. Requests can also add their own security labels (*Break-the-glass* protocol). For example, a doctor may request emergency access from a patient (that does not have permission to access) if the patient is unconscious and at risk of life. These type of labels can have different categorizations (*Confidentiality*, *Sensitivity*, *Compartment*, *Integrity* and *Handling*).

There is a public FHIR server available [53], so it is possible to get an idea of the structure of the data that developers need to work on (see Figure 2.2).


```
"fullUrl": "http://hapi.fhir.org/baseDstu3/Patient/3381806",
"resource": {
  "resourceType": "Patient",
  "id": "3381806",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2018-05-10T16:59:35.209+00:00"
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"> </div>"
  },
  "identifier": [
    {
      "use": "usual",
      "type": {
        "text": "Computer-Stored Abulatory Records (COSTAR)"
      },
      "value": "10006579",
      "assigner": {
        "display": "AccMgr"
      }
    }
  ],
  "active": true,
  "gender": "unknown",
  "birthDate": "1924-10-10",
  "deceasedBoolean": false
},
"search": {
  "mode": "match"
}
```


Figure 2.2: Patient's data (JSON format) [55]

Even though FHIR is still in an early stage of development, there is already interest by some on how to integrate an access control using the resources available in that standard. The official HL7 website has a security information page that includes a white paper available for analysis [50], that gives some insight and possible frameworks for developers to apply on their platforms. But despite all of the useful information on this paper, the access control is always dependent on the type of project that is being worked on and the same solution may not be ideal for different problems.

That is why many parties (including names like the University of Colorado [84]) are still trying to find the perfect solution for this predicament. The main barrier that is often talked about is the policies themselves, where the way that they are constructed may not be fitting for every kind of situation.

There is a high request for this standard and many organizations that deal with EHR want to use FHIR on their systems. By starting to find ways to integrate this standard with existent security protocols and technologies (since FHIR itself does not have any kind of security protocols), not only are we already launching our products and services at the same time of the launch of FHIR, but we are also preventing against initial attacks that are common when companies adopt new products to their systems and they are still trying to fully and safely integrate it. This way, we can prevent users from a system, without the

necessary privileges, to access restricted data.

Some parties are already proposing solutions for when FHIR is finally released as a fully developed product (one of them was already pointed out above, University of Colorado). Hoffman et al. [57] highlight the importance of reporting accurately the time and circumstances of someone's death to discover potential threats and its causes, and help finding possible solutions. But due to lack of means to efficiently expose that data and legislation across different regions, there are still many challenges to fully enjoy this opportunity. Also, some of the current technologies can not completely integrate with some existent web services (the authors give as examples the Clinical Document Architecture (CDA) and HL7 V2). Thanks to the interoperability of FHIR and its vast amount of available resources, all of these problems can be fixed. The authors managed to create an application with a very user-friendly interface where anyone can easily discover a certain Certificate of Death with several distinct information, including a timeline for every diseases and conditions that the patient encountered before its death. All of this data can be retrieved from the FHIR's resources. Hong et al. [59] also focus on the importance of developing easy to read and analyze interfaces that can be integrated with the FHIR resources, since it can take some time and understanding to someone to fully process their structure.

There are even talks on integrating FHIR with more unusual devices. The smart glasses are the main focus of Ruminski et al. [100] and they present the idea of a medical expert using a smart glass and being able to identify a patient (either by facial recognition or graphical markers like bar codes) and having the possibility (through FHIR) of either retrieving data from that patient or inserting data to the records of that patient. A medical expert could also (through his smart glasses) monitor a device that was connected to a patient, check its information (like measurements), upload that information to FHIR and also controlling the device itself.

The authorization aspect of FHIR is also heavily discussed among developers. Some, like Sanchez et al. [104] want to implement an access control through the roles of the requesters. They would do this by developing service that could handle requests and responses. Whenever it got a request, it first would verify the role of the user (requester) and then would check what the permissions associated with that role were. If those permissions allowed the requester to do the operation it wanted then that action would be enforced by the system and a response would be sent to the requester.

Janki et al. [65] proposed using a hybrid model of the access control through roles and attributes, while also adding support for a JavaScript environment. Since FHIR does not have explicit security capabilities (instead, it presents different protocols and models that should be used with policies), two different roles may permit the same general action over a specific resource, but it should probably have some constraints. The authors give an example of a

Doctor and an Assistant that can modify the same record, however only the Doctor should be able to establish and validate the status of that record as "final" (see Figure 2.3). So even though both of them can modify the records of a Patient, that does not mean that one of them can have the same privileges as the other. So the role needs to be verified and also what are the fields that are being affected.

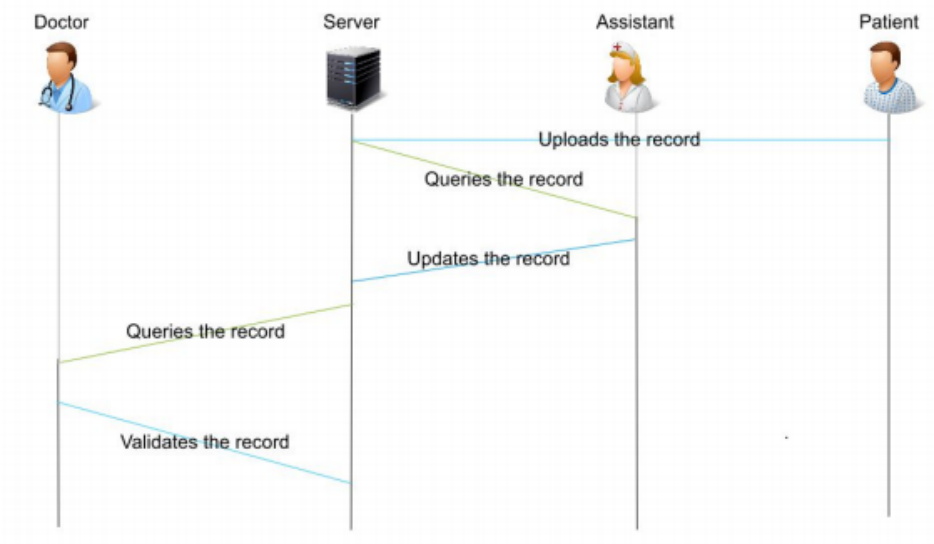


Figure 2.3: Roles in a given situation [65].

With a generalized overview of the advantages and features of using EHR and the FHIR framework, we can start to have a perception of how important they are for a healthcare system and how they can be used and implemented to help experts improve their work and aid patients with the best treatment possible. This will all be useful when planning and developing our system and services.

2.2 Service Authentication

A service can be independent from other services for privacy (e.g. sensible data) or interoperability (e.g. that service can be used with other systems) reasons. However, there needs to be a way to prove that all of those services are who they claim to be and not someone posing as them, that wants to intercept their messages.

Usually there is some sort of authentication, but most of the times it is focused on the service's users (the most frequently used method is the username-password combination) and they usually need to log-in several times in different services and can only access data that is associated to the service they are located in. But, some of those services would also benefit if, for example, another external website could directly access their resources, without needing to exchange classified information (e.g. passwords). Sometimes it is useful for an app (or website) to access resources from a second app, without the need to have both of them installed in the

same place. Especially when the two can use the same resource for their respective services (making the storage of that resource on both apps wasteful).

The main concern with this idea is how are they going to trade that information. To access something protected there is a need to prove your identity for authorization purposes. Should an user give his credentials to one app so that it can gain entry to the resource server? But that may lead to information leakage and put at risk one of the applications.

Thankfully, there are several protocols that can deal with resource management while preventing transmission of private data.

For example, when a service wants to contact another on behalf of a specific subject (access delegation), an authorization framework is required and one of the most famous open standards for that is OAuth [89]. OAuth is an open protocol that was released on 2007, from the collaborative work of Blaine Cook, Chris Messina (both working with Twitter), Larry Halff (from Gnolia, formerly known as Ma.gnolia, now defunct) and David Recordon. It is token-based, in the sense that the main goal is to obtain a unique token that grants access to specific resources. For that to happen, there are several steps to follow:

- The (Service) Consumer obtains a request token from the (Service) Provider.
- The User will be directed to the Provider with the respective request token, so he can authenticate himself.
- If the User is successful, he will be directed back to the Consumer and the Consumer can now trade the request token for an access token at the Provider.
- The Consumer can now use the access token to access (or not) the resources.

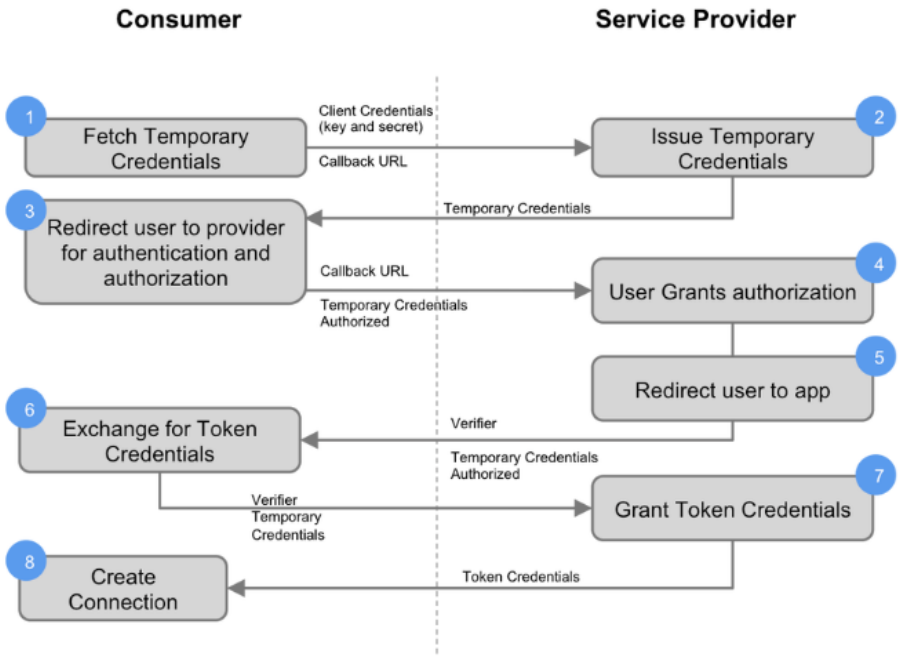


Figure 2.4: Example of an OAuth protocol flow [92].

For security purposes, a parameter with a signature (using a Secret Key) is also sent. The text that is signed needs to be constructed in a very specific way, which can be seen in *The OAuth 1.0 Protocol* page (section 3.4) [45].

Several of these tokens, by default, have time limits for their usage (it can vary from minutes to days), but after obtaining the access token, a user does not need to authenticate again, until the expiry date of that token.

This protocol is often used when external services do not want (or need) to have stored the same resources across multiple services, and rather request that resource from a single service.

An updated version of this protocol has become available since 2010, called OAuth 2.0 [88], where the OAuth Server now validates the service, using a Transport Layer Security (TLS) tunnel [19]. Other improvements include scoping different rights and Revoking rights (after they have been granted).

OAuth is already used in many well-known services and companies like Facebook [38], GitHub [41], Google [43], Microsoft [73], Twitter [118] and the University of Aveiro [119].

Another authentication protocol that was developed in 2005, by Brad Fitzpatrick, is the OpenID [102]. It is an open standard and its main objective is the Web User's authentication and identification, without the need of someone maintaining a service or having, for each service, different names or passwords. When using OpenID, a service can let a third-party authenticate their users, by using accounts they already have (in that third-party authentication server) and then the service can find out the identity of that user through a certificate that the user got after successfully login in the authentication server. It is highly scalable since it does not depend on pre-existing agreements and it allows to have multiple services with the same shared authentication process, including passwords.

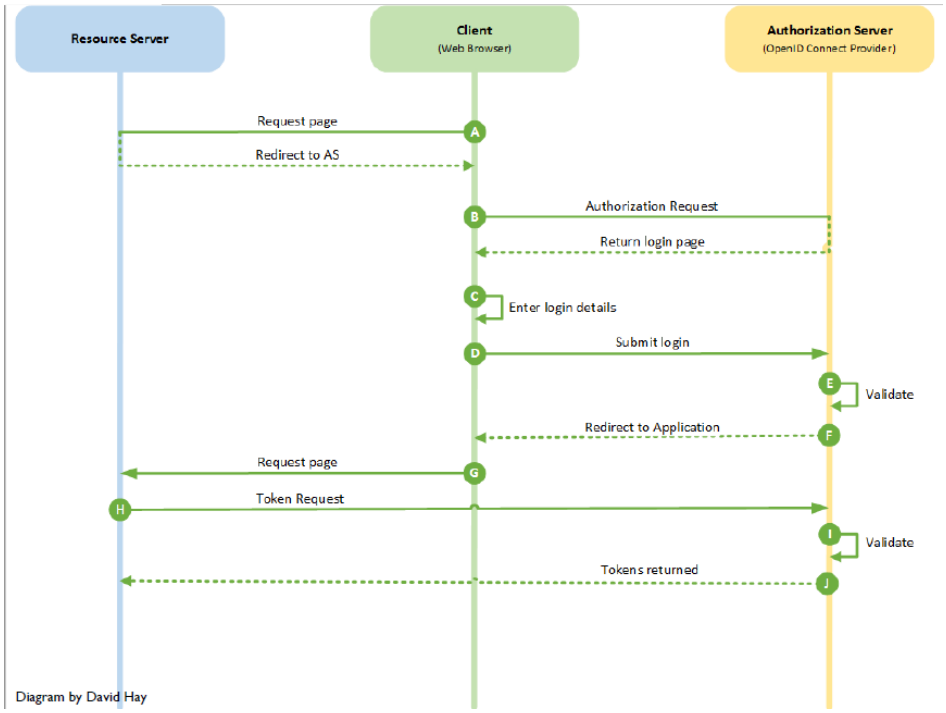


Figure 2.5: Example of an OpenID flow, taken from *M&S Consulting* [75].

This protocol has been used by several websites (AOL, Amazon, VeriSign, Yahoo, IBM and PayPal [18]) since its implementation.

Since 2014, a new standard called OpenID Connect (controlled by the OpenID Foundation) [101] has been built upon OpenID and OAuth. It allows for authentication and obtaining certain user attributes (complementing both OpenID and OAuth). It can also encrypt and authenticate data, discover OpenID providers and manage sessions.

Other commonly used standard that was released by Organization for the Advancement of Structured Information Standards (OASIS) [15] at the end of 2002 is Security Assertion Markup Language (SAML) [81]. It is a standard that focus on the exchange of assertions (declarations of facts, that can be Authentication, Attribute or Authorization Decision, about subjects, for example, an user). It is vendor neutral, it uses XML, Simple Object Access Protocol (SOAP), XMLDSig and XMLEnc and it requires a Secure Sockets Layer (SSL) or a TLS between the servers [82].

SAML is associated with Identity Management, since it deals with identity registration, revocation and termination. It can also handle federated identities (Single identity authority for multiple domains).

The following protocol (Request/Reply) has these steps:

- A Client wants to access the protected resources of a Service Provider (SP).
- The SP redirects the Client to the SAML Authority (or Identity Provider) so he can authenticate.
- After the successful authentication, the SAML Authority sends an assertion to the SP.

- The SP will analyze that assertion and will know if the Client can be trusted or not with its resources.

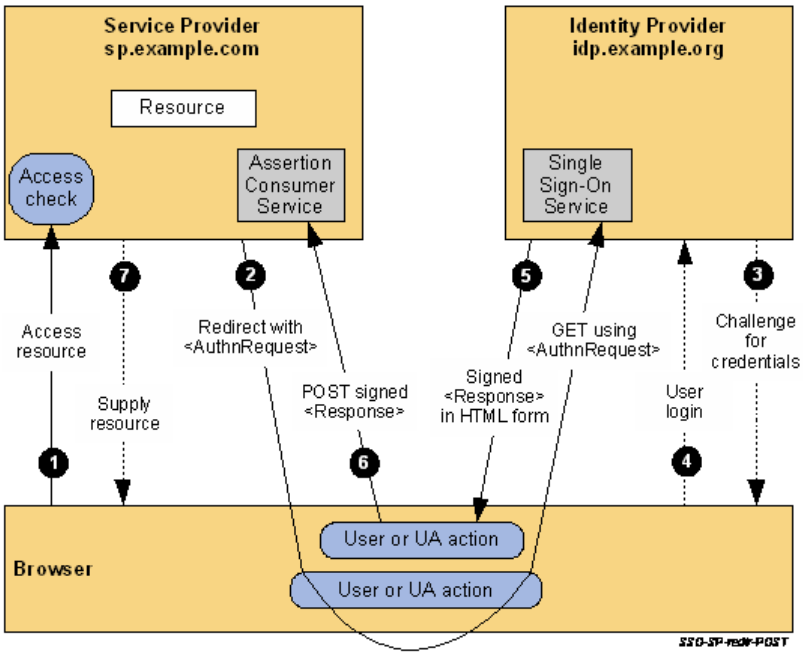


Figure 2.6: SAML work flow [82].

The SAML authentication process can have different approaches. It could be initiated by the SP (Figure 2.6) or by the Identity Provider (IdP). As pointed out above, it is also possible to link several accounts, without the need of someone needing to login with the same credentials over different domains. The log-out process can be done in the same way and if one person checks out of one domain, it can be done automatically in all of the linked accounts.

Even though OpenID is used for authentication while OAuth is more authorization-oriented, OAuth seems the better choice for service authentication, since its main focus is to allow access to restricted data.

2.3 Data Integrity

One of the most important parts of data security is its integrity, especially now when almost any kind of information is available digitally. If someone manages to alter certain information on our system (remove, alter or add incorrect information) they can corrupt several data (or even everything) we have and we can lose evidence of illegal actions and/or tampering.

To (try to) assure data integrity, most developers use well-known techniques. However, most of them are not 100% fail-proof:

1. Data encryption: The main disadvantage is the key that is going to be used to encrypt and decrypt the data (where is going to be stored, what should be the refresh-rate, ...). Maintaining it may be more difficult than needed at, specially if we have multiple services, each one with their own set of keys and if someone can steal them, all of the data may be compromised.
2. Data backup: How many copies of the data are needed? Where should it be stored? What if someone tampers several copies, which copy should be trusted?
3. Access controls: It can prevent outsiders from tampering, but not inside attackers, who want to hide their activities from the system.
4. Input and data validation: It is not enough to avoid tampering of information, since it can be later changed without anybody noticing and very difficult (or even impossible) to find out.

Another commonly used method is through the use of timestamping. Whenever we need to confirm that a certain operation was done or verified at a certain time, using and adding timestamps values can prove to be useful for future analysis. Timestamps are normally depicted as long values that represent a time instant in milliseconds and can be converted to the respective date format (example, the timestamp 1527514030 refers to the date "Monday, 28 May 2018 13:27:10").

Seeing that this document will focus on logging services and that we want to make it possible to exchange logging information between other services and validate the integrity of that data, timestamps can be used to indicate that a certain service validated something in that instant. Additionally, if the Trusted Timestamping process is implemented, not only can we say that one service validated that data at that time, but that service can not deny that it did that validation, since it signed it (see Figure 2.7).

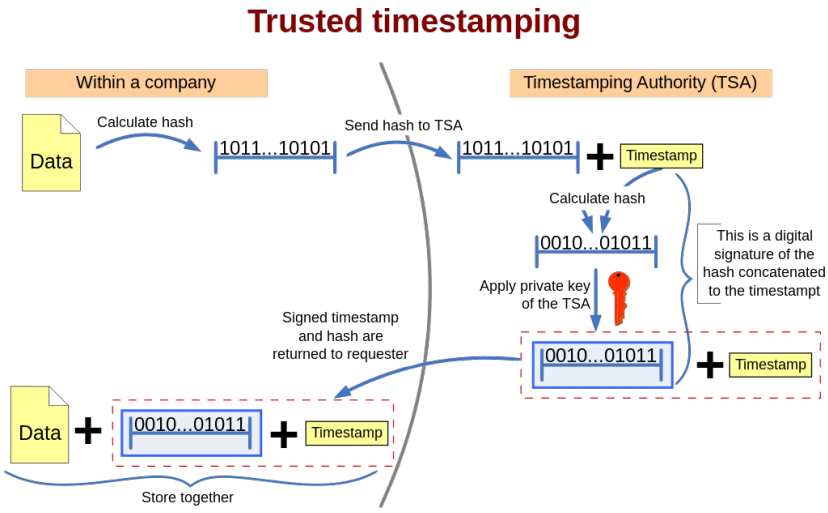


Figure 2.7: Trusted timestamping [124].

However, this may not be enough, because the data sent may already be forged and/or

the one that did the timestamp signing did not even verify the information correctly. Data integrity is something most of the software developers try to assure. There are some established methods for that, but there are also some new (and underused) techniques that can avoid certain problems that the old-school security tricks have. One of these most recent techniques is blockchain.

The details of the blockchain's origins are scarce. All that is known is that a person (or group) by the name Satoshi Nakamoto developed it in 2008, so it could be used on Bitcoin (a digital currency) [13] and solve a common problem with digital money that could only previously be fixed with the aid of a trusted authority, the double-spending dilemma (where a digital token could be replicated and used more than one time). Since its inception, blockchain has been mostly integrated with cryptocurrencies. Bitcoin is the most popular (and the first decentralized) cryptocurrency that uses this (blockchain's) public ledger. It was released in 2009 and its value has been over the thousands of dollars since early 2017 [23]. Because of Bitcoin popularity, many alternative cryptocurrencies were released (Ethereum in 2015 [37], Ripple in 2012 [98], Cardano in 2017 [21], ...). Other proposed markets where the blockchain technology could be applied are the insurance industry (e.g. to verify the existence of an accident), the healthcare industry (e.g. electronic medical records) and even the Internet of Things.

The basis of the blockchain's architecture is quite simple. Like the name implies, there are several blocks that are linked with one another. Inside each block, we can have several attributes, but normally we have the data itself, a timestamp, a hash (a unique identifier or a digital fingerprint) and the previous and/or next block hash. These hashes are the key attribute to confirm the integrity of our data, because each block is dependent on the next and/or previous and if someone wanted to tamper one block's information it would need to tamper every single existent block (which would make the intrusion very easy to detect).

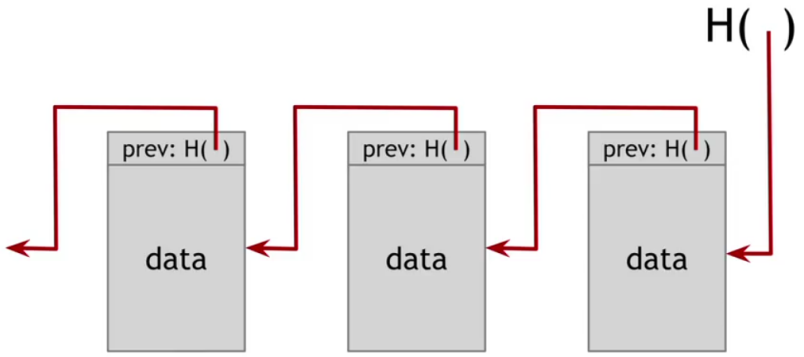


Figure 2.8: Simple blockchain Architecture [77].

The blockchain also differs from other data storage implementations by not having intermediaries services or participants that handle each service's data and are the only ones to have all of the system's information.

Instead, there is a Peer-to-Peer (P2P) network, in which every participant has access to all of the system's data (shared ledger), and contributes to the data's validation. That means that every single block must go through each system's node before being inserted into the shared ledger.

There are two different ways that the blockchain's P2P network can work. It can either be permissioned or permissionless. The difference between the two is that while in a permissioned blockchain, every participant is uniquely identified and there are policies to constrain network participation and access to certain information or details, in a permissionless blockchain, all participants have the same rights and they can access anything.

When using a blockchain, there is an assumption that every participant does not fully trust each other. With that in mind, there is a need for a distributed consensus mechanism for the validation process. There are several methods for that, but there are three main ones, which are *Proof of work*, *Proof of stake* and *Proof of elapsed time*. *Proof of work* means that, for a block's validation, the amount of computational resources spent by the network's node is put on consideration. When there are nodes with different influences on the system, then a *Proof of stake* mechanism can be used, in which we prioritize the validation of the better-influenced participants. If time is an important factor, the *Proof of elapsed time* mechanism is the best option, since every node must wait for a specific (previously assigned) time, before one can of them can claim validation leadership (this claim is influenced by the node's amount of resources contributed to the network).

One of the many advantages of using a blockchain is the possible integration of smart contracts. Like the name suggests, these are contracts that do not require the trust between the involving parties. It is basically an agreement or a set of rules that are automatically enforced by computer protocols.

Many parties are showing interest in using the blockchain technology outside of cryptocurrency and there are already many articles and reports explaining how that can be implemented in other markets. Of course, sometimes, changes to the blockchain's structure are needed to be able to make the process more efficient for different areas of application:

"A Prototype Evaluation of a Tamper-resistant High Performance Blockchain-based Transaction Log for a Distributed Database" [4] proposes to use blockchain to maintain the integrity of the log files of a system. However, there can be some time and energy consuming operations in order to apply that technology and that can lead to many slow updates and insertions on the log file system. The authors recommend using two different layers for the blockchain architecture (2-layer BlockChain (2LBC)) to avoid these kinds of situations, where the first layer is responsible for fast operations and the second layer is responsible for more heavy operations and ensures data integrity.

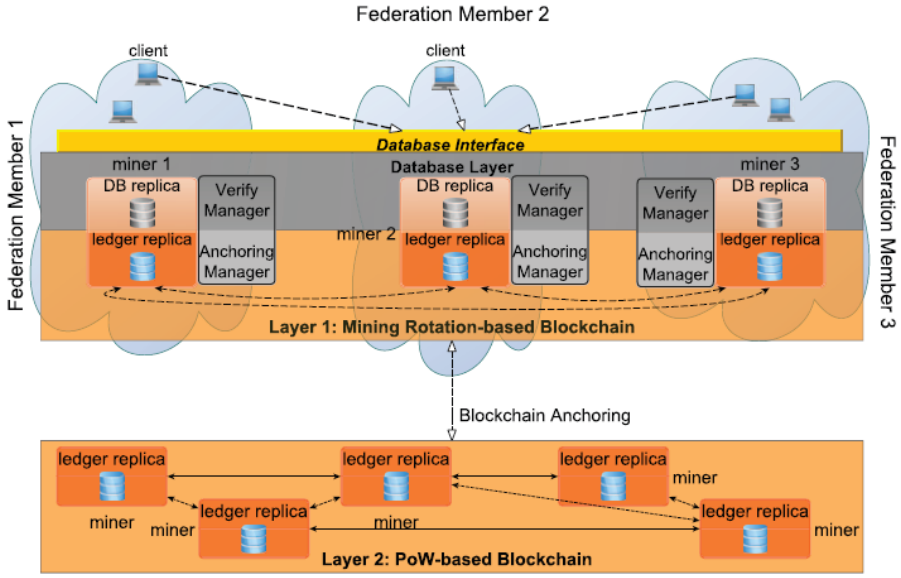


Figure 2.9: 2LBC architecture [4].

The author of the article "Blockchain: solving the privacy and research availability tradeoff for EHR data" [69] suggests, to avoid privacy invasion, using the blockchain technology for Health Information. Using all of the blockchain's features, it is possible to keep someone's health data secure and only allowing the essential information to be retrieved by someone with the authorization to do so.

Although the main focus of the blockchain is data's integrity, the article "When Intrusion Detection Meets Blockchain Technology" [72] also proposes adapting this technology on Intrusion Detection Systems (IDS). This could help solve two of the major problems with IDS, which are Data Sharing and Trust Management (mainly caused by the lack of trust between participating parties and fear of an insider attack).

Since most of the (data integrity) approaches are outdated, blockchain seems the best alternative to use since it can store any type of asset (transactions, logs, properties, ...) in a way that is reliable, cost-consistent, efficient and secure. Adding the signed timestamp, we can prove that our blockchain was certified at that point by that service, by verifying that signature. The more services that can do this procedure for a specific blockchain, the more robust it will get.

2.4 Logging Applications

A platform, application, or website, for consistency and security purposes, must keep the records of the different requests and accesses inside their system, to monitor, detect and prevent certain problems from malicious users/programs and to improve their systems. So, storing logs (the events inside the system) in a secure way and also maintaining their integrity

is an essential feature for any system, so that later they can be retrieved to perform an audit (analyze and study that data), to check if the policies and regulations are being followed and there is not any type of illegal actions.

Storing those messages in their raw format inside a file is probably how simple solutions are implemented. However, by only doing that, we will have a very disorganized and unnecessarily complex log file, making the tracking and analyzing process much harder to execute. Thankfully, there are some workarounds to solve these problems.

One of the most commonly used methods to generate detailed events so that they can be stored (usually in a text file) and later tracked is the Python standard logging module [94]. With this module, we have at our disposal several functions that can help us have consistent and categorized logging records. For example, it is possible to control the different levels of the messages (in ascending order of severity: DEBUG, INFO, WARNING, ERROR and CRITICAL) and eliminate the unnecessary ones. It is also possible to display the date and time of an event, format the structure of the messages to our liking and even create and associate different loggers to different events and messages. Even though this module allows us to construct a very detailed and organized log file, it is still vulnerable to tampering as someone that could access the file could erase evidences of their activities on the system.

There is also a similar logging functionality for Java [87], that has almost the same functions as the Python module with some minor differences (for example, we now have seven logging levels: FINEST, FINER, FINE, CONFIG, INFO, WARNING and SEVERE). Unfortunately, we will have the same integrity problems as with the Python one.

Over the years, there has been several applications that handle the logs of a system, so that a developer does not have to worry about the creation and monitorization of the logging records. Retrace (an Application Performance Management system) is a product from Stackify [114] that stores all of the log data on a centralized location with many available features (error tracking, performance analysis, detailed log searching, ...). Still, it does not solve the tampering problem, since all of the information is stored in the same place and there is not a way to verify if someone modified any of the data. Other log management tools suffer the same problems (LogRhytm [90], Splunk [12], Logentries [58], ...) because their main concern is to store data and retrieve them quickly and efficiently, without worrying about possible integrity problems.

There are several alternatives to handle the events that occur on a system, however, most of them do not have a solution that fully prevents and detects non-authorized tampering of log's data, that can have proof of illegal activities inside a service.

There are some research works that talk about innovative ways to develop a logging service by integrating it with existing standards and technologies. Huh and Martin [61] focus on Virtual Machine (VM) and recommend deploying the logging service into an independent and isolated environment (a VM) and also to follow the same logic for the log storage and other

possible logging-related services (like log access services or log analysis manager services, see Figure 2.10). By following this architecture, each service is totally isolated from one another and if one is compromised, the entire system will not be affected. And since every service has a specific role, it is harder for an attacker to tamper data, unless he can compromise every independent environment where the logging-related services are and it also makes it easier for the logging service to verify the integrity of the data before storing it to the respective storage center.

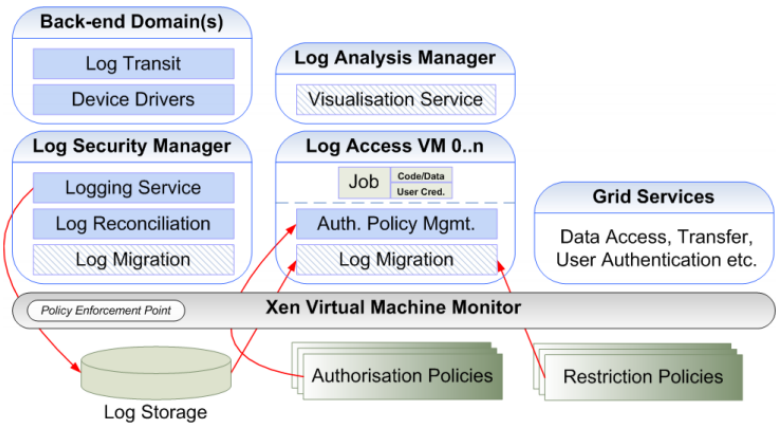


Figure 2.10: Abstract View of Trusted Logging Services [61].

There is also a paper that proposes a solution that eliminates the incompatibility issue between different log formats across different services. Huemer and Tjoa [60] present a solution that creates an interoperable LOG environment between different services, by creating XML-enabled LOG files. One of the advantages of following this logic is that since most of the logging formats are dependent on their order and delimiters, by using the XML format we remove that dependency. Additionally, since this format is widely used, there are many tools available (and also free) that can protect and parse the original log format to the XML one and with the usage of XML-schemas inconsistencies could be easily detected. Interoperability is also present (even when different services use different log formats) after they parse the messages to the XML format.

Another approach to the problem presented in the earlier article [61] is discussed by Sukmana et al. [115], but they focus more on cloud computing. These authors [115] refer that many Cloud Service Providers have their own logging format and since these providers store data that comes from the same user or company, log aggregation and analysis can become complicated. They propose a unified logging system that comprise of a log collector (to store log files periodically), a log duplicate checker (to verify repeated entries, which some Cloud Storage Providers may occasionally have) and a log format normaliser (to normalise the different log files from the several Cloud Storage Providers). Only the fields that contain important information and that are present on all Cloud Storage Providers should be used on

the normalisation process (for example, the Uniform Resource Identifier (URI) of the request is present on three Cloud Storage Providers, but all give different names to the field, so we need to retrieve the respective value and associate to the field of our unified log). This would create a consistent and uniform log system that had the capacity of integrating different Cloud Service Providers.

Another interesting concept is the Web log mining, where through the processing of the web server logs, it is possible to discover and analyze user's patterns. With this information it is possible to update and improve websites, according to the user's behavior. Singh and Meenu [110] recommend using the Web Log Expert Lites 9.3 (a Web log analyzer tool) [2] since it allows to discover the number of hits, page views, visitors and bandwidth. With these statistics, a web analyzer can know the activity rate by hour of day and find out what are factors that can be improved to increase the visitor count.

Wireless Telecommunication Systems can also benefit if they analyze their logs (which are normally discarded), since it has the bigger portion of the generated data. However, they can contain too much information which can affect the the analysis of logs. Chen et al. [22] propose first processing the log entries using Natural Language Processing, so that they can be analyzed with a Machine Learning algorithm and create training models and later compare new entries with the trained model and detect unusual behaviors.

Logs can even be useful to detect and improve the Quality of Experience (QoE) of a network's user as discussed by Sawabe et al. [107]. Many network users not only want their network everywhere, anytime and quickly deployed but also they want to browse faster and not wait too much to watch what they want. This can be improved by the operators with the analysis of the communication logs. However, due to the delay of the HTTP sessions and users, over a few moments, accessing several web pages and since estimating Web QoE involves estimating web sessions from those HTTP sessions, sometimes those estimates are wrong. The authors propose a method that can estimate the correct number of web sessions through the relation between the number of HTTP sessions and the number of hyperlinks (assuming that the number of pieces of content of a web page is approximated to the number of hyperlinks) and thus improving the accuracy of the Web QoE.

Even the military can benefit from logging. According to Kim et al. [68], exchanging log files (that contain the status of the naval combat system) can help improve viability and combat capability, and since it is very demanding to gather that information from the naval combat system, it is proposed creating a software that can gather, store and analyze those logs to reduce the time cost of obtaining the pretended (filtered) results.

Investigators like Alimbuyog et al. [3] want to solve a problem that happens often in their country. In the Philippines, there are frequent motorcycle-related accidents. And they suggest

using logs to gather evidence of those events and find out what caused it. Those logs will come from several components of the motorcycle and the helmet used. The helmet has a camera and an alcohol detector, while the motorcycle has several ultrasonic sensors, brake detector, gyroscope, led indicators and other elements. The helmet will send its information through bluetooth to the motorcycle which in turn will store it in a integrated SD card. Even though this is still in an early development stage, the prototype works and the data collected was valid for motorcycle accident analysis.

Similar to Alimbuyog et al. [3], Ghandi et al. [39] also see the benefit of logging, to help people that use wheelchairs. Since different wheelchair designers have different ideas of how their products are going to be used (some assume that people only use wheelchairs off-road or only for small distances), additional data can help these designers to create a product that serves every type of user needs. By integrating several hardware components (including a memory card) into the wheelchair and storing data from both the wheelchair's usage (through the speed and bout length) and the terrain where it is being used and then sending it to a cloud server, this information can be analyzed and used to help designers adapt their wheelchairs to benefit different types of clients, according to their needs.

Logs not only are useful to check the system's behavior and pinpoint when it started to malfunction, but they can also detect and prevent threats from attackers. Dahmane and Foucher [27] warn that outsiders are not the only attackers that exist, but that also that these threats can come from the inside. This can turn into a very dangerous situation, since these attackers have (or previously had) authorized access to the system and can tamper or steal information. Sometimes, this can even happen accidentally (malicious vs misuse). However, using the logs of a system, it can be detected whether or not there are strange behaviors and potential attacks. The authors [27] advise using the Markov Chain model for each employee of a company. In other words, we can have the patterns of activities of a worker for that day, like if normally he does something out of his working hours, if he often connects or disconnects devices or his web browsing usage. We can even know which order he would normally do these activities (first he logs in, then he connects his device and later he uploads something). All of this information can be retrieved from the logs. Then, we can compare the different datasets of activities of the employees with a threating scenario and conclude if it is a usual or unusual behavior for that person.

Of course, there are other models and techniques that can be used to detect irregularities in a system. For example, by using the k-means algorithms, specifically the K-nearest neighbor classifier (proposed by Gao et al. [40]), where through analysis of the web logs, it can be detected deviations from some user's routine, through the calculations of its entropy (the bigger the entropy, the higher is the possibility of being a malicious user). The usual behavior of a user will be represented by a set of points in space (example, X-Y space where each axis represents a different behavioral characteristic), that will form a cluster and whenever there

are new activities from that user, new points will be added accordingly in that space and if the those new points are too far away from the center of a cluster, something out of the ordinary may be happening.

Cao et al. [20] even proposes a two-level machine learning algorithm, where first, through a decision tree classifier, there is a selection of the log files (that are labeled as either an attack or normal behavior) and then, using the hidden Markov model, a normal data set is modeled, that can help detect strange occurrences and spot unknown attacks. This algorithm was tested with (real, from an industry) 4690000 log messages and there was an accuracy (of detection) of 93,54% and a false positive rate of 4,09%.

Malware can also be detected through logs, as explained by Deli et al. [31]. This can prove to be useful to regular Internet users that fall victims to these malicious software that can replicate itself across the system and steal information (among other actions). By analyzing and processing the logs and encountering unusual behavior (the author recommends using the Hadoop framework for this, since it can handle Big Data), these software can be detected and even with some detail (for example, if it is a virus or a spyware). A framework by the authors is presented in Figure 2.11.

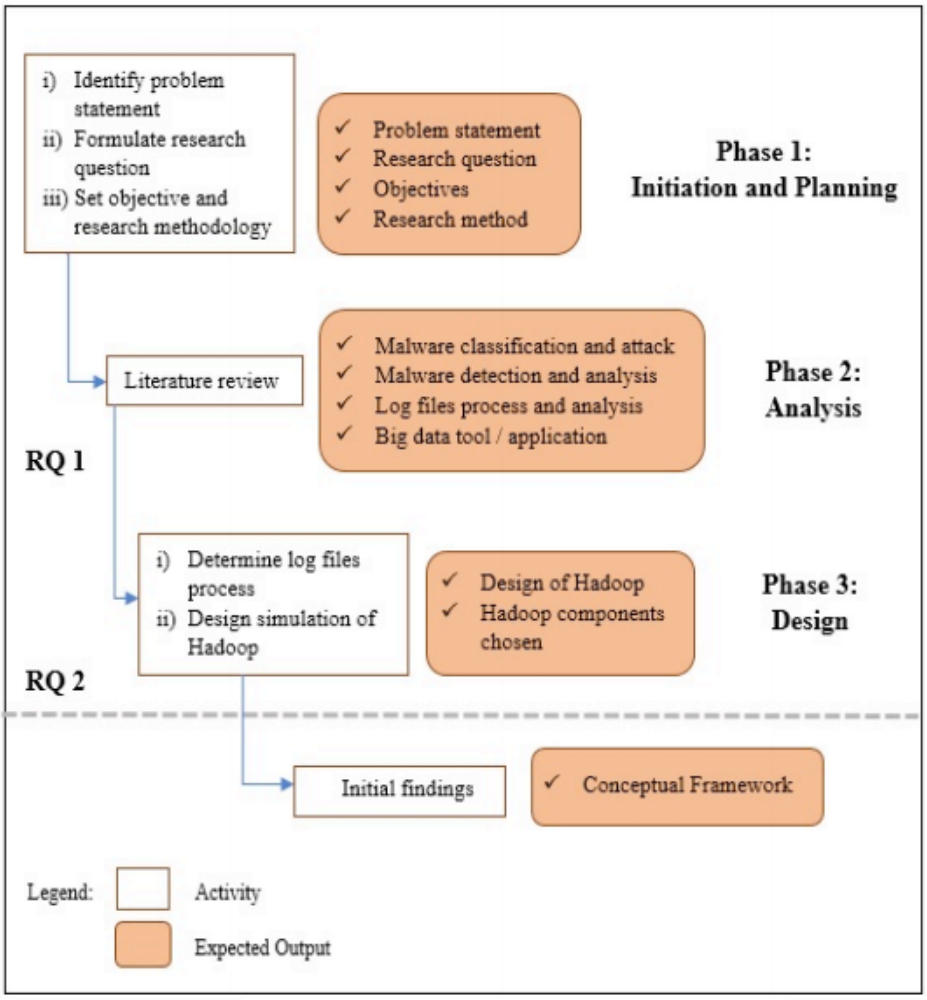


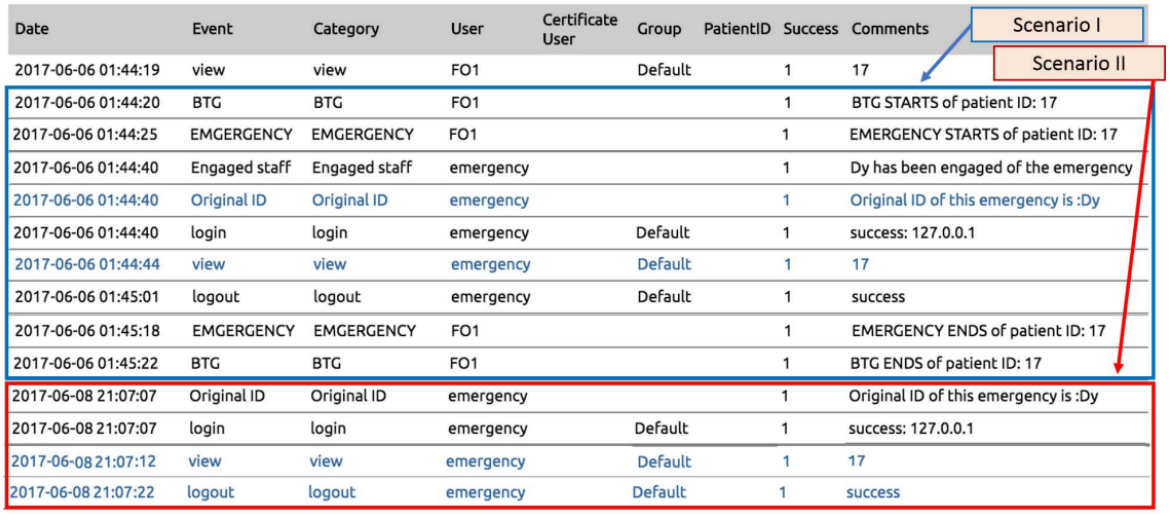
Figure 2.11: Operational Framework [31].

This type of attack' detections through logs can prove very useful to the Internet of Things (IoT), since it deals with the network communication between several devices and could be easily exposed by attackers. Ahmadon et al. [1], similar to Dahmane and Foucher [27], propose a solution that compares the current log events of a system (that can have several devices attached) with the flow of events that are supposed to occur (only this time they use a different model, the Data Petri Net) and check if there are any violations. This could help detect either malfunctioning devices or cyber attacks and knowing that with IoT we will have devices like traffic lights, medical equipment and heavy machinery connected to a network, this could prevent life-affecting problems.

Banks are also another institution that can use logs for fraud detection. As Rahmawati et al. [95] point out, in almost 100 countries, there were losses of 1.4 billion (of US dollars), caused by 1,388 fraud cases. These authors see the opportunity on finding a solution that can detect these cases early, using the event logs of a bank. They propose using the Hidden Markov Model algorithm, that can predict (with the respective probability) the sequence of events

(states) inside the system. Through a sequence of log events (that contain several elements regarding the activity, timestamps and users involved) and using its different elements, if the value of probability of fraud is bigger than the value of no fraud (regarding the current state in which the event finds itself), then there is a possibility of a fraud occurrence. With some tests, the authors, using 90 different bank scenarios (where 10 of them were fraud cases), they accurately predict 94% of the total outcomes.

Wickramage et al. [123] identify a problem regarding how logs are created in some health care systems. The *Break-the-glass* situations, where a doctor can access data that he is not authorized to during an emergency, is not always for the greater good. Sometimes, it is not to save a patient's life, but instead to sell that data to someone else. Even though that access may be registered into the log storage, not much information is recorded (for example, the user of the access request is identified as "emergency") and that makes harder to track who did what. The authors propose a simple solution, where those systems should include additional details like logging information and emergency situations. With this additional data, it is possible to build a timeline where it is possible to filter who was the person that access the private date and what were the reasons (example, a doctor checking a patient's private information after a emergency event occurred with that patient). Figure 2.12 shows the timeline of aggregated events that allows for a more in depth analysis of two different scenarios, where scenario 1 represents an acceptable case of *Break-the-glass* (a doctor checking a patient's data after he had an emergency event) and scenario 2 represents an unacceptable case of *Break-the-glass* (a doctor checking a patient's data without any critical event happening at the time).



Date	Event	Category	User	Certificate User	Group	PatientID	Success	Comments
2017-06-06 01:44:19	view	view	FO1		Default		1	17
2017-06-06 01:44:20	BTG	BTG	FO1				1	BTG STARTS of patient ID: 17
2017-06-06 01:44:25	EMGERGENCY	EMGERGENCY	FO1				1	EMERGENCY STARTS of patient ID: 17
2017-06-06 01:44:40	Engaged staff	Engaged staff	emergency				1	Dy has been engaged of the emergency
2017-06-06 01:44:40	Original ID	Original ID	emergency				1	Original ID of this emergency is :Dy
2017-06-06 01:44:40	login	login	emergency		Default		1	success: 127.0.0.1
2017-06-06 01:44:44	view	view	emergency		Default		1	17
2017-06-06 01:45:01	logout	logout	emergency		Default		1	success
2017-06-06 01:45:18	EMGERGENCY	EMGERGENCY	FO1				1	EMERGENCY ENDS of patient ID: 17
2017-06-06 01:45:22	BTG	BTG	FO1				1	BTG ENDS of patient ID: 17
2017-06-08 21:07:07	Original ID	Original ID	emergency				1	Original ID of this emergency is :Dy
2017-06-08 21:07:07	login	login	emergency		Default		1	success: 127.0.0.1
2017-06-08 21:07:12	view	view	emergency		Default		1	17
2017-06-08 21:07:22	logout	logout	emergency		Default		1	success

Figure 2.12: Log files timeline [123].

As previously discussed [115] cloud computing is a big thing and Sridhar et al. [113] warns to the dangers of user's data being misused (either by external attackers or internal system

administrators) and proposes a solution where these users can easily interact with their logs in a simplistic and informative way and get notified about potential problems. Using OpenStack (a cloud operating system that allows users to monitor their activities on the cloud) [86] as a basis, the authors developed a program that filters the most useful information out of the vast amount of data of a log file and presents an interface that allows even users with low-level knowledge of a log's structure to detect possible problematic situations (like log in attempts during hours he was not available to do so).

Some works also highlight the importance on finding solutions to prevent tampering of the log files, so that certain activities do not get hidden forever. Medeiros et al. [71] alert to these type of attacks and the authors also warn that even techniques that do not allow change over data, if an insider attacker has physical access to the disks, the data is still vulnerable. They also express that the main objective of any secure log scheme should be to assure that any tampering made should be detected when the log file is verified. The solution proposed is a strategy where the new entry's payload should be similar to the previous ones. The payload can have an identifier, the log message and its parameters and a timestamp. Then the identifiers between two payloads can be compared and if they are the same, the most recent payload changes the value of its timestamp to the time difference between its original timestamp and the previous payload timestamp. This will create a chain of logs that are linked through values calculated between their different payloads. For an additional layer of security it is also advised to use symmetric encryption mechanisms.

Other authors propose alternatives to guarantee integrity in log files, like Hartung [46] adding sequence numbers (to detect reordering attacks) and epoch (the date and time relative to the timestamp) markers to the log entries while Crosby and Wallach [26] recommend using a Merkle binary tree, where the events are stored into leaves and permits the reconstruction of past version and past commitments (sent to auditors). Ning et al. [79] propose, for vast amounts of logs, a concurrent authenticated tree data structure, that would greatly improve the performance of the system. This structure is similar to the Merkle tree, but it supports concurrency and it also uses chameleon hash functions (this gives the possibility, to prevent collisions, since it would be almost impossible to someone to discover the same function value, without the knowledge of the trapdoor key that was used for the original value).

In the same way that these authors thought about new ways of integrating protocols to a different type of service, in this document we will follow a similar path by integrating two existent techniques into a service that normally is not directly associated with them. While some techniques used on these papers will also be used here, some modifications were made to accommodate the goals that we are trying to achieve and that fit better in our work than the other methods.

2.5 Auditing through Logs

Data should be examined with some frequency, to find out if protocols, policies and rules are being followed correctly or to find out the cause of a problem. When someone perform this type of action because of those mentioned reasons, that means that an auditor is performing an audit. An auditor can be someone from the inside or the outside of the auditing target and the auditing itself can be done to several items that contain information like accounts, documents or books. An audit should be divided into several stages in which should be included the planning, gathering of information and its analysis. In the end, the results should lead to efficiency improvement or indicate that everything is going according to plan (or even better).

Auditing something may have different procedures to follow and relevancy on different areas and industries. But in the health care industry it is a very vital process for a health professional to know if a type of treatment of a certain patient is working and if not, what are the causes for that failure.

This type of audit normally follows a pattern that results in a cycle. First, there needs to be a preparation, where the standards (topic, resources, criteria, ...) are defined. Then the auditors will evaluate and compare the results of the experiments with the ones that were supposed to be obtained and (if necessary) they will adapt and change their methods and repeat the whole process until the desirable outcome is attained.

The more detailed the process is, the higher is the probability on finding the factors that are causing the problems. Problems can be found in the most mundane attributes like the address or age group or on more important ones like type of blood or gender.

This auditing may help teams of medical experts to create plans or adopt different procedures to try to help patients. For example, the Kikuyu Eye Unit team from PCEA Kikuyu Hospital [67] used the auditing process and were able to know which type of patients needed more attention and care for cataract operations, since some patients with different problems had different outcomes after the surgery.

With logs not only is it possible to see and evaluate the activities of a user for security purposes (check if he tried to log in several times in short period of times or attempted to access great quantities of data on a few seconds) but additionally, health experts can study the different behaviors of their patients (for example, by checking the patient's sensors activities) and add those standards and results on their audits.

There are some papers that try to explore more about this topic (some of the papers discussed on section 2.4 are related to the auditing process). Zhang et al. [126] advise on using XML Access Control Markup Language (XACML) for both authorization and auditing procedures. Every time a user makes a request to the service to access something, the service will first determine if that subject can access that resource via an exchange of messages with the *authorization decision point* and then the service will communicate with the *auditing decision point* to verify if the auditing for that request is possible. If all of this is permitted, the service will log that access and send it to the appropriate auditing service or storage

center and will finally send and enforce the decision to the user's request (if the data access request is authorized than the service will send the requested data to the user).

Other papers consider the privacy of the information present on some logs. The article Samavi and Consens [103] explain that it is important to verify that all of the services that handle the same data are following the rules and protocols previously agreed among themselves. To guarantee that compliance, the authors propose a model (called Linked Data Log to Transparency, Accountability and Privacy (L2TAP)) with several components. Every time there is a new log entry, the logger will send that event to the L2TAP and it will be constructed to the appropriate (L2TAP) log format. This format has several characteristics and values associated (like `eventParticipant`, `eventData`, `logTimeline`, ...) and it will be stored so that later the participants of the log can be registered and link their privacy policies to that event. If they want, the data subject (the user that is referred in the event) and data sender (the service that has the event) of an event can add more privacy preferences and policies (for example, the requester that ask for that event must obtain the consent of its data subject). This privacy preferences follow a similar structure to the XACML rules. With all of this steps followed, whenever there is a data requester asking for a particular event, it needs first to ask for its access and verify if there are any obligations to perform. If there are and it follows them, then it can access that data (see Figure 2.13).

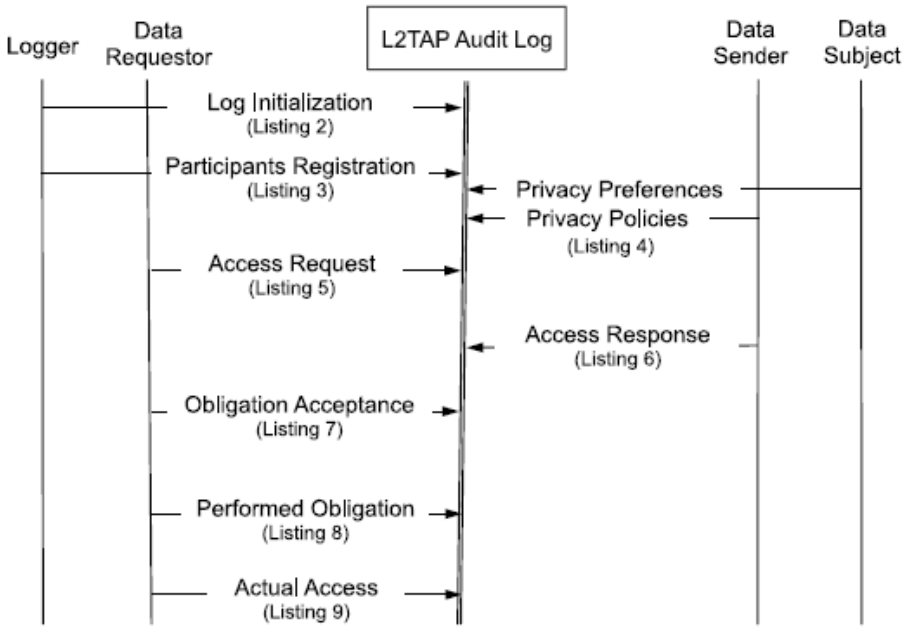


Figure 2.13: Sequence diagram visualizing event occurrences in an L2TAP log [103].

Similar approaches have been discussed and presented, showing the importance that people give to the auditing process. Hasiloglu and Bali [47] talk about a central audit log with an Application Programming Interface (API) manager, where whenever there is a request to

its web service (identified in the API manager) there are several steps to follow before the desired response is given (examples, the Internet Protocol (IP) of the request is verified and an authentication is required from the requester application). Whether the request is successful or one of the steps failed, this request will be also logged with all of the relevant data, to detect unusual behaviors.

Integrating emerging technologies with existent services is also something developers see as a promise solution. Suzuki and Murai [116] present the idea of using blockchain as a communication channel, where every message is sent through the channel and also stored as a block in the blockchain there. Every other server or client must search for these blocks and wait for the ones that are addressed to them, so they can retrieve them, analyze them and possibly respond (once again, that message is sent and store into the channel). This proposed scheme is represented in Figure 2.14.

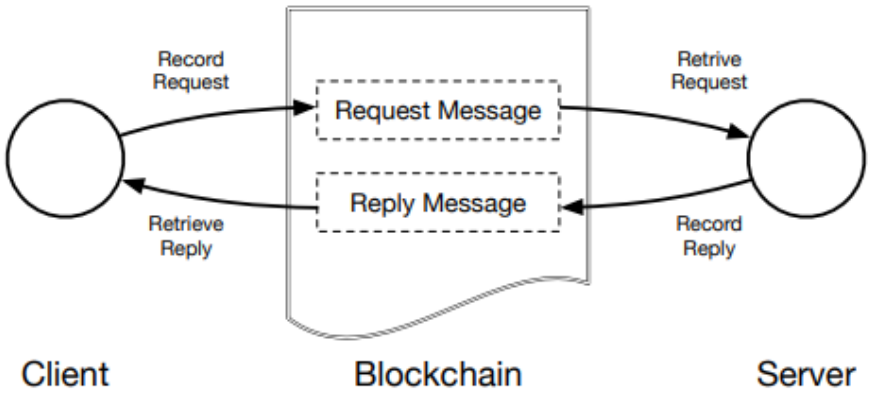


Figure 2.14: Blockchain proposed scheme [116].

However, it should be noted that this is still an early concept and not every proposed solution benefits best for different type of systems and certain modifications must be done to fully take advantage of these type of technologies. This will be better discussed further into the document.

2.6 Access Control

Resources being accessed or exchanged inside a system is a common practice, but some resources should not be available to everyone. Sometimes, they contain sensible or important data that could be wrongly used. So, it is important to define what can be accessed, how and by whom.

One of the most basic solutions is to follow the "Everything/Nothing" strategy, where someone can have access to all of the information or none at all. This could be done during the login process, where there are administrator passwords that grant access to all of the system's data. However, several fields on a specific data may contain private information that

should only be seen by a distinct person or group of people, which eliminates the usefulness of this simplistic approach.

The best decision would be to develop a decision mechanism (with the appropriate set of policies, rules and restraints) where the subject's access level can be evaluated. This can be done via a certain set of attributes (XACML [83]) or by the role of a user (JSON Web Tokens (JWT) [10])

XACML stands for XML Access Control Markup Language [83] and was released in 2003 by the OASIS organization [15]. It is a policy language that can be used to describe who is allowed to do what. Since it uses XML, a language that is both simple and flexible, it can be used (almost) anywhere. There is not also the need to code authorization policies in each service, since there will be a centralized location that will handle that and it can even specify in the messages sent to the services what they should do.

XACML uses the Request/Response method:

- The Request message will ask about the permissions associated with a specific subject.
- The Response message will answer the request with a permit, deny, indeterminate or not applicable.

The architecture of the centralized location that will deal with the XACML messages should follow the same structure of Figure 2.15.

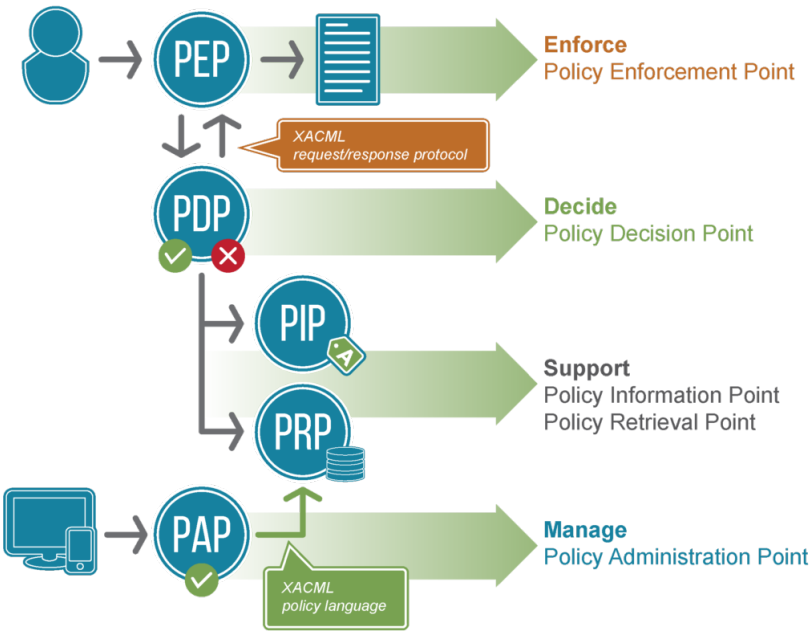


Figure 2.15: XACML Architecture Flow [76].

Every component on Figure 2.15 has an important role. The *Policy Decision Point (PDP)* evaluates the requests and issues the decisions about an access, the *Policy Enforcement Point (PEP)* makes the requests and enforces the decisions with specific methods, the *Policy Information Point (PIP)* provides the information about attributes and values, the *Policy*

Administration Point (PAP) manages the policies and the *Policy Retrieval Point (PRP)* is the system that stores the policies.

One of the incentives to use this language is how meticulous someone can get in creating the rules (if someone wants a rule that is only valid during weekends, they can use a function that allows for that. Figure 2.16 shows an example where someone can only login from 9am to 5pm). This rule can then be inserted into a Policy, that can have several other rules and has a policy combining algorithm associated (example, if any rule of that policy returns a *Deny* (for a specific request), then the final decision is *Deny*). There are also *PolicySet* that can have one or more rules.

```
<!-- Only allow logins from 9am to 5pm -->
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
      <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
  </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
      <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
  </Apply>
</Condition>
```

Figure 2.16: Rule time period restriction [80].

It is up to the rules administrator how precise he wants a policy or rule to be.

Since XACML is attribute-based, it is possible to categorize the values that are going to be sent and analyzed to reach the decision. There are 4 possible types of values that can be used, the `subject`, the `resource`, the `action` and the `environment`. Every one of them can have multiple values and they can also be used when creating the rules so they can be directly compared to the respective attribute used on the request.

XACML has been implemented in several authorization services, such as ViewDS Access Sentinel [121], NextLabs Control Center [78] and Security Policy Tool [64].

There are also several articles that show the benefits of implementing XACML with different type of services (a example was already given in section 2.5 [126]). It was already tested following the health sector regulations of Saudi Arabia, as seen by Atiq and Alsulaiman [8], where they give some examples where XACML was successfully integrated with the Ministry of Health of Saudi Arabia [106] protocols (examples, allowing the disclosure of private information for treatment purposes or to help authority forces regarding a crime committed by the patient).

To reduce the workload that a service that uses XACML has while searching for its several policies to find the rule that can have the response necessary to answer a request, Zhang and Zhang [127] use a decision inheritance tree, that will assign different priority levels to the

possible policy paths (Figure 2.17). The higher priorities should be assigned to the policies where the rules had more responses for the requests, since it would greatly improve the time that would normally takes when it is not clear where that rule could be and it would be necessary to check every path blindly until it could be proven that that subject could access that resource. It is not always going to be accurate in the first path, but the more tests and attempts are tried, the higher will be the accuracy. Another authors, to reduce the workload mentioned earlier, advise on using caches for the PDP and PAP (Ilhan et al. [63]) where the request and its respective access decision and finder result pair are cached to improve the efficiency of future requests, while others (Kateb et al. [66]) suggests splitting a policy into several ones, where each one would be handled by different PDPs instead of one.

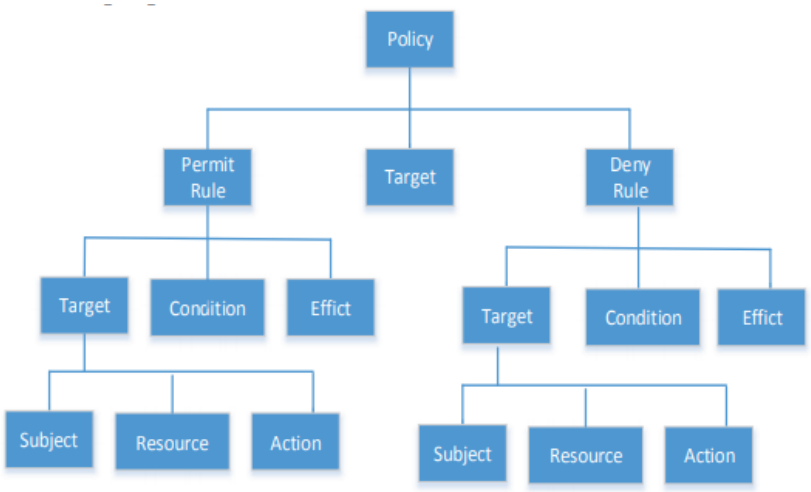


Figure 2.17: Decision inheritance tree [127].

However, certain services, features and data only need the subject's roles to be accessed and JSON Web Tokens [10] are commonly used for access controls that only use roles to decide if an authorization request can be approved or not. JWT is an open standard that allows to create access tokens (in a JSON-based format), where claims can be included. With this, for every user that exists inside a system, a JWT can be created where several attributes can be added, like its role, identification, access level, specific actions it can execute and much more.

The token itself is divided (with periods) in three parts:

- Header: Contains the hashing algorithm and the token type (`jwt`, by default).
- Payload: Includes all of the claims, associated with the token's entity. Those claims can be registered (not mandatory, but can prove useful, since they can have information such as `issuer`, `Issued At`, `expiration time` and more), public (defined at the *IANA JSON Web Token Registry* [62]) and private (custom claims, used between trusted parties).
- Signature: Using the header and payload (both encoded), a secret and the hashing algorithm mentioned in the header part, a token signature can be generated. The secret (for example, a key) used for the signing process is chosen by the interested parties.

Due to its (JSON) format, it can be used in almost every major programming language, like Java, C, .NET, Python and JavaScript.

Knowing that there are two types of access control (Attribute-based Access Control (ABAC) for attributes and Role-based Access Control (RBAC) for roles) and since it is important that the authorization for a resource can be done in a simple way and that it can be used with any system or service (interoperability) in a way that external services can be involved, XACML would be the ideal standard to use.

CHAPTER 3

Auditable Logging of FHIR Services

Chapter 2 introduced several security protocols that are commonly used in almost any kind of systems. However, which one is the best alternative to apply to a product is reliant on the system's objectives.
For that reason, this chapter introduces and explains what are the goals of the project that is associated with this document and proposes a solution for a Logging service with auditing and access control features (that solves some of the problems mentioned on the previous chapters), including the requirements, architecture and use cases that will help make that product more secure and flexible to use and interact with other type of services.

3.1 SOCIAL and FHIR Ecosystem

The Social Cooperation for Integrated Assisted Living (SOCIAL) project is supported by COMPETE 2020 [93] (which main goal is to reinforce the competitiveness of the Portuguese economy and its presence in the international market) and focuses on the development of technologies and innovative information services to support centered care networks on the society and focused on an age group above 65 years old.

The SOCIAL platform is being developed in order to benefit the communication, information sharing and provision of services between the users (mainly people on the senior age group) of the platform, and their respective carers and people of interest (doctors, family members, political managers, ...).

Some examples of the possible scenarios (regarding the usage of the SOCIAL's application) include the log-in process from users or someone who can benefit them with their participation on the system, like doctors or family members that are responsible for their well being, to help them access the features of the system, the possibility of creating alarms, whenever a user requires help (for example, the user is feeling very sick and feverish and creates an alarm, so someone can come over its localization and help), the ability to query certain data (private or not) of a user, that can help handling certain operations with more precision (for example,

knowing that person's blood type to quickly aid them with a blood transfusion during an emergency) and also the capability to make appointments (or other interactions) between users and their doctors, in a quick and easy way.

With the examples presented above, there are certain aspects that need to be considered. In order to comply with the current legislation, these objectives raise several questions, regarding the access to applications, the authentication of stakeholders (professionals, politicians, and support citizens), the necessity to create *break the glass* mechanisms, which will allow users to access certain records and data that they are not supposed to access during emergencies, as well as ciphering mechanisms of databases containing extremely personal information. It will also be necessary to develop activities that allow the analysis by certified auditors.

One of the requirements of the SOCIAL platform is the possibility to integrate it with the Fast Healthcare Interoperability Resources (FHIR) standards framework. As it was explained before (Chapter 2, section 2.1), FHIR [48] was developed by the HL7 organization [109] and provides resources [49] that permits the exchange and management of several Electronic Health Records. Since it can be used among different systems (thanks to its interoperability features) and it also can be implemented into service-based architectures [52], this will allow SOCIAL to perform several required actions, like to create, modify and request data that will be useful to improve the efficiency of the platform and its users (for example, if a doctor adds new clinical records of a Patient through the FHIR server, every other service that uses this server will also have these new updates instantly). However, FHIR by itself does not guarantee that these resources can be accessed by only authorized personnel, but it provides infrastructures that allow to develop an access control system [50], through the usage of *Security labels*, that contain information such as *Context of Use*, *Data Sensitivity* and *Control of Flow* [51].

3.2 Requirements for the Logging Subsystem

Any system that wants to protect itself from (inside or outside) attacks, should be able to create, process and store logs, for all of the different events that occur inside it, through the implementation of a *Logging* service. But, if someone could access them, they could manipulate them in order to hide their activities and making impossible to detect anomalies or fraudulent events.

This could lead to serious problems on the SOCIAL platform, for example, some doctors could eliminate their queries of patients' data (to hide unethical and abusive accesses), perpetrators could delete their failed attempts at trying to log-in as someone else or someone could make another person look like the culprit by editing or creating an illegal event. The integrity of these logs is very important, because even if the authorized person is accessing the events of someone else, that information will not be useful if it was tampered by an attacker. The blockchain technology (as previously explained on chapter 2, section 2.3) can guarantee that integrity, thanks to its hash field and the link between the different blocks of the chain. Using a stream (for example, the concatenation of the data of the logs) that follows a certain

pattern (for example, the logs of a service, sorted by their timestamp), we can use a hash function with that data, and get a fixed size value, that can only be obtained with that specific stream. That hash value will be added to a block (alongside other information, that will allow to re-calculate that hash, like the identifiers or timestamps of the log events used) and if there is any block that was created previously to that one, it must be updated, so it can reference this new block (the same can be done vice-versa, where the new block references the previous one). This can be done, for example, by creating a field that has as the value the hash of the other block. Following this logic, every time a new block is added to the chain, the more robust and tamper-proof the blockchain will be, since changed log events can be easily detected through the corresponding block's hash and even the integrity of that block can be verified by the other linked blocks' information. It is also important, when there are multiple services (each one with their respective events), that the validity of those logs are legitimate and not someone trying to sabotage or pretending to be that service in question.

However, this should not be the only concern to worry about. On almost every platform (at least the ones with private data) there is a need to know who (normally a system's user) is requesting what (usually a resource), so that it can be decided if he is allowed to access it or not. If there is not some mechanism that can manage those decisions, then anyone can access everything and nothing will remain private for very long. Because of this dilemma, developing a ABAC for a system that deals with restricted information is essential so that it is possible to allow access through attributes (they could be of any type: `subject`, `resource`, `action`, `environment`, ...) that are used by policies.

Some may say that using only a RBAC is sufficient to restrict data from users. However that is not recommended, since RBAC only assigns roles or privileges to subjects (so that they can access certain features from a system) and there could be situations where users may have the same roles or status as others, but they should not have access over the same resource (for example, someone with the role "Doctor" can access more features than a "Nurse", however not every "Doctor" should access the same data, only the data from their respective patients). However, having both of those Access Controls could be beneficial to a system and assure robustness and consistency, where the RBAC could be used to know which features of the system the subject can access and the ABAC for knowing which resources he can claim or write over. And since the FHIR standard allows the integration with these type of authorization components [50], this can be easily achieved using its resources.

In our case, the private information will be the log events and the ones trying to access may be a service or a person, that want to perform an audit with that data. Some of the access rules may be defined by us and/or by an exterior component. Other private data is stored in databases, but this is out of scope for this work.

Like it was explained previously, by not restricting the access to certain resources, privacy will not exist on a system. And when projects like SOCIAL handles great amounts of private information of many users, there could be dangerous consequences if there is not a safe way to deal with that data. We could have situations where someone could (illegally) check the

activity of a user or a malicious user could know a sensor's activity from another user and conclude if they are at home or not. It would also be easy to create ransom situations, using that data as leverage.

If there is a need to have an authorization mechanism on a system, an ideal solution would be the implementation of the XACML. With that standard, it would be possible to make decisions, regarding the access to certain resources, in a way that (its use) would be accessible by any service on the system.

The logs themselves, contain data from the events that occur inside the system. Each service inside the system will send those events (authorization requests, log-in attempts, user's accounts modifications, sensor's levels variation, ...) and the respective user linked to the respective activity, to the *Logging* service and it will process them accordingly, so that they all have the same structure and so it can be possible to build activity flows (either of a service and its events across different users or of a single user across different services), even if they originate from different services with different log formats.

However, since they come from several services, some events can include sensible and private information. For example, as it was mentioned earlier, if we have a "Sensor detection" service, that sends to the *Logging* service information about the activity of the different objects of the user's house (doors, windows, ovens, ...), someone could know the day-to-day routine of that user and sell that information to someone else (it could be a journalist or even a burglar).

By integrating FHIR with the log access process, it is possible to develop an auditing system for medical experts (and other types of FHIR resources, like organizations or related people) to verify different timelines (constructed from the log events) for their respective patients and verify possible errors or problems with their methods. And since FHIR has resources that show the relation between two different subjects, we can guarantee that the requester for that patient's logs is indeed their doctor and not someone else without the authorization to do so.

Knowing the technology and framework that are going to be integrated with the *Logging* and *Auditing* service, it is necessary to define what are the requirements necessary, so that they can take full advantage of the blockchain and XACML features. First, all of the services should have mechanisms that allow communication and transmission/reception of different data type objects. They should be able to process requests, analyze them and build the appropriate response and send it to the respective destination. It would also be advisable that each service had its own Database Management Systems, to store keys, passwords or other information for the possible authentication and signing/verification operations and also to store the log events and blockchain data, to have a more decentralized database and to guarantee consistency.

The *Logging* service should have its own storage center (the Database Management System (DBMS)). Every time there is a request where it is necessary to store or retrieve data from the DBMS, the *Logging* service will do all of these operations (including update, create and delete operations). It should never allow access to it to anyone else, because it could

be misused (example, given wrong or repeated values across different entries) or internally attacked (delete entries by purpose) by another service. The DBMS itself should be able to store all of the necessary information clearly in such a way that is possible to filter and retrieve only the necessary information quickly and efficiently. That information includes the log's data (like the event itself and the user that triggered it) and block's data (the associated log's metadata, for example, the identifier of the first and last log of that block and the respective hash value for all of those events). All of these requirements for the DBMS are very important for the *Logging* service, because we need to store the logs that the other services send as fast as possible to avoid inconsistencies on the respective stream of events and we also need to recover that data to build the blocks and then store them in the same DBMS. The better divided the data structure is inside the DBMS, the better the data filtering is and the consequent data exchange performance.

There should be also a *Timer* service or program, to send the alerts for the block's insertion process and for other possible time trigging events. Instead of having a service or person sending these alert requests, it is better to have an automatized process to do that. This module does not need to be convoluted, it only needs to send messages to the *Logging* service (it does not even need to have the ability to receive them) at certain periods of time, with the possibility of easily changing that time frame. Having the block insertion process being triggered by time it is better than being triggered by the number of events sent to the *Logging* service because we can have periods of time without many of these events and the insertion process is only triggered after a long period of time has passed and risking compromising the still to be created blocks. In the case that during a period of time, no new event has been stored into the DBMS, if the block insertion process is triggered, no new block will be created.

Because the *Logging* service communicates with different services (either they be inside the same platform or not) that want to send their logs to be stored, it is crucial to provide different types of logging features according to each service rules and protocols. Some may want to build the hash themselves or want to give their confirmation before the *Logging* service adds a new block to the blockchain, while others do not want to be involved in the logging process and just want to send and receive logs without wanting to be involved on the logging operations. This is very important to define early on, because it promotes flexibility inside a system, specially if there are services that cannot share certain parts of the data and need different approaches on how to handle their logs in a secure way.

The logs itself may come in different structures according to how they were generated in each service. Those services should not need to change their logs to be the same as the *Logging* service, since it would affect their performance and also because they could be integrated with several other projects that already work with those type of logs. The *Logging* service should be able to parse them into the appropriate structure that needs to be able to have a uniformed log structure across several different services. The developers may choose to add

methods that handle this parsing or using already available parsers.

A similar strategy should be applied to the communication between the different logging services, that will share the `Logging` service's data (distributed database). Even if the different logging services trust each other, it does not mean that they are willing to share everything. Due to policies and/or restrictions they may have to exchange distinct information. One logging service could send only the metadata (the block information) to another service so that it will not know about the associated events or could send the metadata and the corresponding events to another service so that it can have access to all of the data.

How the other logging services handle the received log events and verify the integrity of the blockchain must be established a priori. Either the service validates everything blindly (not advisable) or needs to know how the hash of the blockchain was generated. For the hash option, the service needs to know what was the original data mapped, to be able to reconstruct it and compare its value to the one of the block (of the blockchain) associated. It is not necessary to send that data in its raw format, if the logging service knows how to construct it from the already received log events (for example, by concatenating every log timestamp by ascending order). If the logging service already received the metadata (the block's data) it can immediately compare the hashes and store it (or not) on its DBMS, if he still does not have the metadata, it should request it from the one that send the events and follow the steps previously mentioned. Then it should send a message to the log event's sender with proof (example, a signature) that shows that it verified that block and that could be used as irrefutable evidence of that validation.

Now, we are going to focus on the auditing aspect of the service, since it is an important process for any logging service, that allows to help and improve the efficiency of the system and certain users. Whenever there is an audit, it is assumed that the auditor has good intentions in doing so, however, in some cases, that may be not true, and someone could be trying to access log events from our DBMS for malicious reasons. That is why it is important to add a data access component to the *Logging* service, to verify who can and cannot access what. Since we will be integrating the XACML framework with this data access component, we can reach that decision through the analysis of the several available attributes (like the identity of the requester, its role, the identity of the resource, ...). Through the development phase of this component, it is important to first determine certain aspects. For example, building the policies should be always reliant on the goals that are trying to be achieved, but since we know the intentions of the SOCIAL project, the flow and structure of the policy's creation could be established in a somewhat generic way, for services that handle EHR.

First, every policy may have several rules and each rule must follow a pattern. In our case, a rule is associated with a subject and that can be divided in sub-rules, one for each role that the subject can have over different resources (example, the same person may be the "family doctor" of several patients and the "nutritionist" of different patients). So, every rule has only one subject and each sub-rule can have multiple resources and an action associated to it.

If we follow this principle, we must define which attributes must be mandatory or optional. In our case, we will establish as mandatory the `subject` (that can be represented by their different identifiers), the `resource` (also represented by different identifiers) and the `action`. As optional, we can have the `environment` attribute (that can be used to define what services it can access).

One of the main features of using XACML is the ability to use functions for every attribute inside a rule. For example, when a subject has time period for accessing a certain resource, every time a new request arrives, its timestamp can be compared with the timestamp of the respective rule.

Other example on the advantage of using functions with XACML is being able to associate several values to the same attribute and then verifying if the request received has any of those values for that attribute. If a doctor has different identifiers for every hospital or institution he works on, that can be very useful, since we will not need to create the exact same rule for the same doctor to access the same resources, multiple times, for every different identifier that he has. By using the function "string-at-least-one-member-of" and "string-bag", we can create one single rule for a subject access to certain resources, as long as one of its identifiers is present ("string-at-least-one-member-of") in that array of values ("string-bag").

All of these XACML features will be essential to handle more complex requests and to provide very specific responses to almost any kind of situation. Those requests and responses should not be exchanged directly between the PDP and the service that requested the access, because to provide interoperability and to avoid adding unnecessary layers of complexity inside the other services, the PEP must be translator between those two entities.

As explained before, the PEP handles the service's requests and they should be in a data type that can be used by almost any kind of system. In most cases, the PEP operates with request contents in the JSON format (example, {"`SUBJECT`":["FHIR – 221712"],"`ACTION`":["read"],"`RESOURCES`":["FHIR – 1667158"], "`ENVIRONMENT`":["service_authentication"]}), since it can be implemented with Java, .NET, Python, Ruby, PHP, Node.js and Go. By using values in array format, it is possible to send a request with multiple identifiers for the same entity or for different resources entirely. In the previous example, the Subject is identified as 221712 (its FHIR' ID) and is trying to access data (the log events from the authentication service) from the Patient 1667158 (FHIR ID). In this case we will say that the action is "read", because the subject wants to analyze that user's activities on the system.

Now, the PEP will transform that JSON message to a XACML message, where the PEP also added the *Environment* attribute with the value of the current timestamp) and will send it to the PDP. When the PDP receives the PEP's message, it will compare it to the rules of the policy file. If the rule has the same field's values than the message, the PDP will send a *Permit* message to the PEP. Otherwise, it will send a *Deny* message. It is also possible to add more complex responses, but for these scenarios, the *Permit*/*Deny* answers is sufficient.

The PEP, after receiving the response message, may have to transform it to the appropriate format and send the answer to the service that made the request.

Another issue that needs to be focused on is how and when should the policies be updated. Is it better to be periodical or by request, should all previous rules be deleted when there is a new update or should they always be preserved and the new rules be added to the existing ones and should the rules be constructed automatically or by hand?

As always, it is subjective. For every different scenario, one option may be better than another or a mix of both could be the best solution. One hospital may want to follow a protocol that is common among all of the other medical facilities, but they could also want to have some exceptions regarding the accesses of their staff, so the best solution would be to build automatically (example, retrieving data from a server) the policies and with the added option to a administrator to add or edit certain rules to obey the protocols of that specific hospital.

An institution may want the policies to be routinely updated every week with the choice of being able to request for updates outside of that routine and they could also want to maintain old rules (for persistence purposes) for a certain period of time (in which they will later be deleted). Fortunately, the XACML is flexible enough to handle all of these different variables and restrictions. It can be easily modified to attend any need and it can handle more complex access regulations.

3.3 Attacker Model

To better understand how using blockchain can guarantee the integrity of a stream of events, consider the following scenario. An attacker wants to delete an event that identifies him as the one who performs an illegal activity (or modify the event, so that someone else is shown as the culprit). He succeeded in figuring out how to access the *Logging* service's DBMS and erased the event. However, that tampering can be detected when re-calculating the hash of the stream of events that the event was part of and comparing it to respective block's hash. The only solution that the attacker thinks about is changing the block's hash, by re-calculating the new hash and exchanging it with the old one. But, the previous block has also a reference to the next block's hash, so the attacker also needs to replace that block's value. Additionally, if we previously used, as the original values for the hash cryptographic function, the concatenation of the respective events plus the hash of the previous block, then the attacker would need to change every block's hash from the blockchain to completely hide its tracks, exposing himself to intrusion detection. And even if that attacker would not mind being caught (as long as it succeeded on the attack), since the blockchain technology is (normally) integrated with a distributed database, one entity could always ask for the original values (the ones from before the tampering attempt) from another entity that previously exchanged that information with (Figure 3.1 shows a attack model for this type of situation).

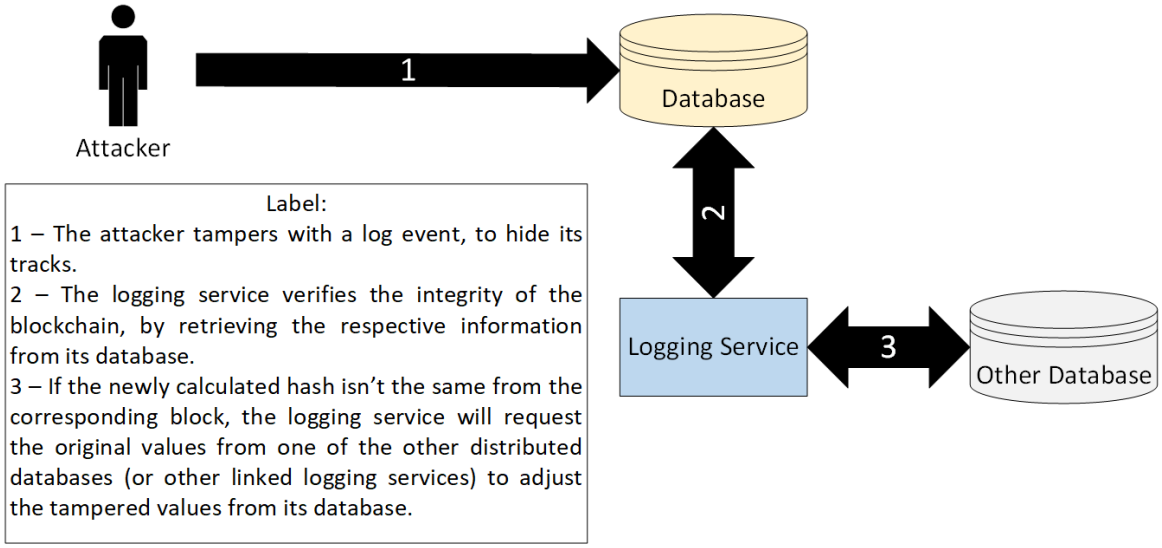


Figure 3.1: Log event tampering attempt

Even if an attacker manages to tamper the other systems' databases (as shown in Figure 3.2), if signatures were exchanged every time new data was sent and stored in a different database (for example, using the trusted timestamping process described in chapter 2, section 2.3), those signatures can prove if that information was modified or not (since the original information is needed to verify a signature) and that it was indeed the system that signed that data. If that is not verified, the information must be retrieved from another database, and if every system's database signatures can not be verified, then the information must be restored from somewhere else (for example, an off-line storage, that stores a snapshot of the system, every hour).

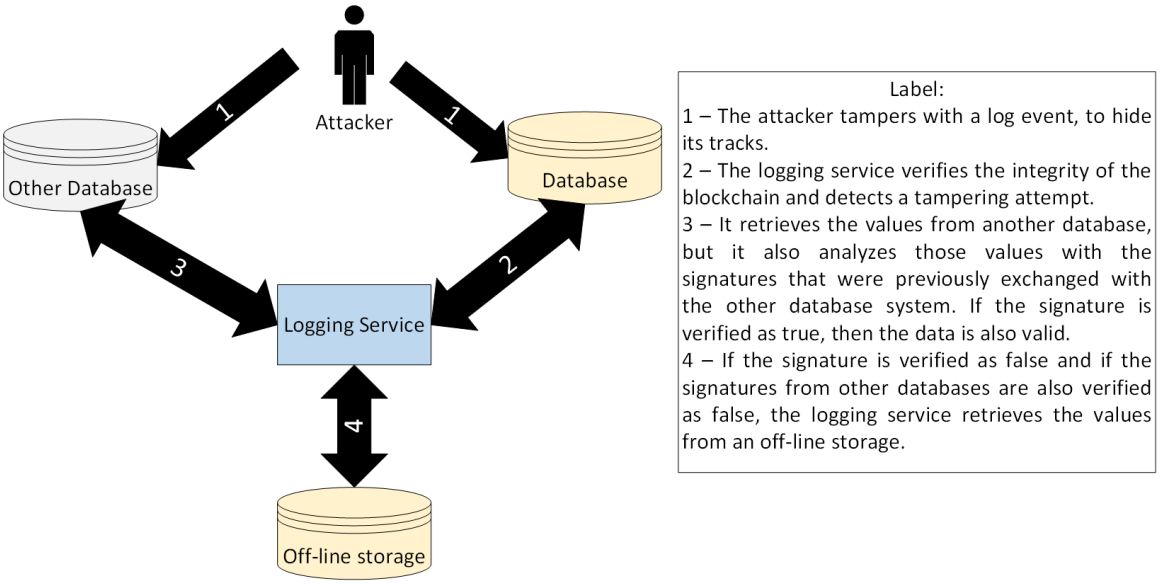


Figure 3.2: Log event tampering attempt

3.4 Proposed Architecture

Knowing the different requirements of our *Logging* service, now we must conceptualize it before starting to develop it and how it can integrated into the SOCIAL platform, so that the different associated services may exchange data and communicate with it. From a high level perspective, it could be something like the architecture demonstrated in Figure 3.3.

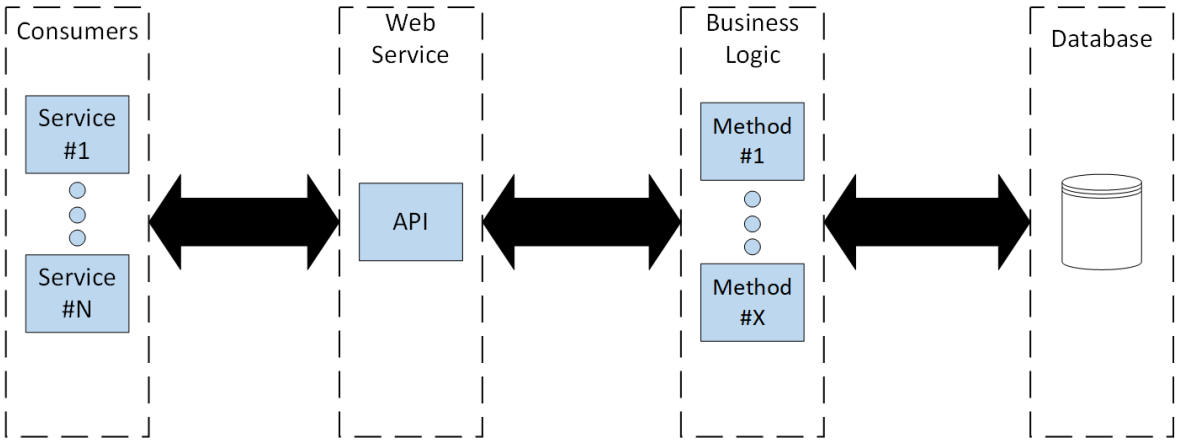


Figure 3.3: High Level Service Architecture

In this diagram (3.3), we divide our service into four layers. From left to right, first we have the consumers, that are the services (the requests from the users of the system will be forwarded by the respective services) that send and request logs (the access control may be used for certain cases) and other logging services that exchange information with our service. Then we have the service layer (web service), that has the Application Programming Interface (API) that the other services use to communicate with our *Logging* service and access our features. Next it is the application layer (business logic), which is where the code libraries are and the service processes and executes the requests it receives from the previous layer. Finally, we have the data layer (database), that houses the DBMS that the *Logging* service uses to store or retrieve information.

As discussed previously, the blockchain technology could be used to guarantee the integrity of the logs and the XACML standard could be used to restrict accesses to those events. To be able to take advantage of the blockchain and XACML properties and features, the architecture where the *Logging* service is inserted should follow the diagram depicted in Figure 3.4. In this more detailed architecture based on Figure 3.3, we can see that the application layer now shows how the different components mentioned in the previous section (like the access control and timer) can be integrated with the standard *Logging* service and that the service is the only one that communicates with the DBMS (that contains the log's information, as well as the blockchain's data). The *Logging* service component shown in this architecture is also the one responsible for handling the auditing operations.

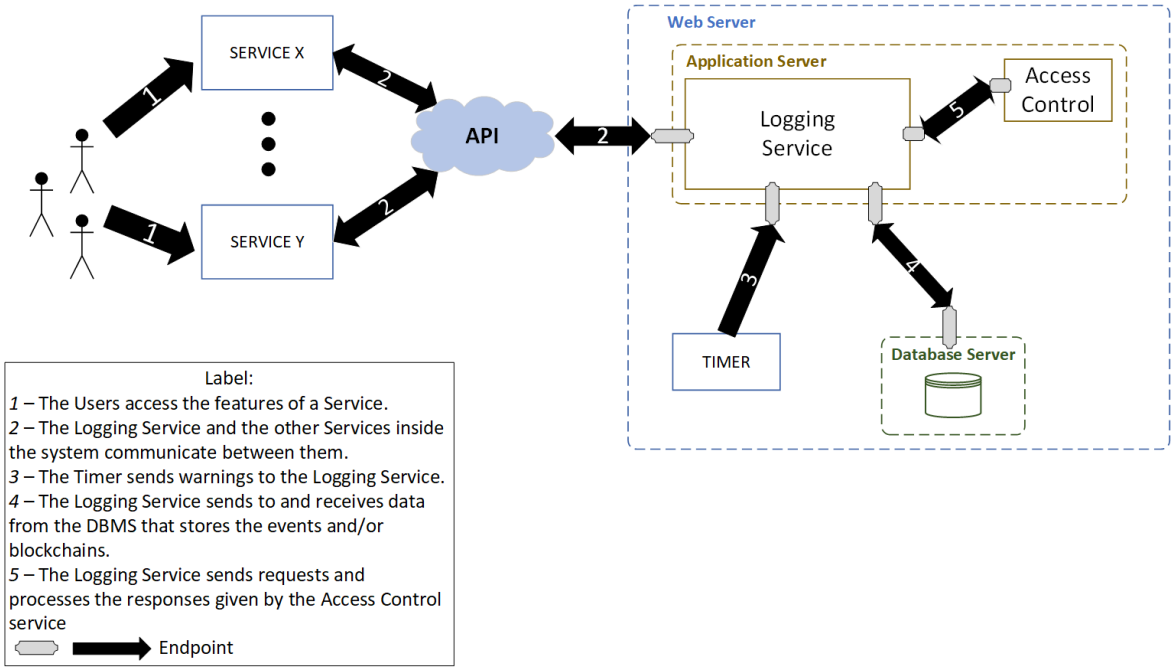


Figure 3.4: A more detailed *Logging* Architecture

Following Figure (3.4), whenever there is an activity on a service of the platform or system, that data will be sent (by that service) to the *Logging* service. After receiving and verifying the legitimacy of that message, the service will construct the appropriate object/document to deliver to the storage center. Whenever there is a need to analyze any data stored (auditing process), the *Logging* service should be able to handle access request messages, send them to the access control and analyze the decision responses. Depending on those responses, it should be able to filter specific results (precise fields, like names and dates) and by a specific order (sorted by name, date), either ascendant or descendant.

After a certain period of time (or by an authorized request) the *Logging* service will be alerted to start the new blocks' insertion into the respective blockchain. After confirming the insertion of new events since the last block inserted on the chain (or this could be the first block ever on the chain) it will calculate the respective hash and retrieve the necessary data from the storage center (for example, the last block's hash, to insert that value into the new block's previous hash field). After having all the required fields filled, it will insert that block to the respective chain and update the next hash value from the last block inserted.

The integrity of the chain can be verified anytime to check for any inconsistencies. Since every block of the chain has fields like the identification of the first and last event, we can recalculate the hash of all the events occurred between those two checkpoints and check if it is the same of the one on the block's hash field. The next and previous hash fields on every block can also verify if the block itself was not tampered too.

Since we are integrating XACML with the log access control, its architecture should follow the one presented on Figure 2.15 (chapter 2, section 2.6). The access control only interacts (directly) with the *Logging* service, has shown on Figure 3.4. Inside the XACML architecture,

the most relevant components are the PDP and the PEP, since they are the ones who will handle the decisions, while the other components handle the policies. If we want our system to be accessible and flexible, so that other services can easily interact with it, we should not ask them directly for the XACML format message, that should be the PEP responsibility. The PEP must consume a simple format message, to promote the interoperability between the different services and systems that may want to communicate with it and transform it to the respective XACML format. Then, it will redirect it to the PDP, that will evaluate the message (according to its policy's rules) and reach a conclusion. That answer will be sent to the PEP, which will transmit that information to the one that did the query.

For security reasons, only the *Logging* service should directly communicate with its DBMS and access control, to avoid the misusage of these components from others. With that in mind, having all of these elements inside the same (web) server is the most ideal solution, since it makes the transmission of messages faster (which is important when handling great quantities of information) and helps prevent users from other servers to easily manipulate those entities. Since the timer component is not an application and most existent DBMS have their own (database) servers, only the *Logging* service and access control need to be deployed on an application server.

It is also important to focus on the log and block (from a specific blockchain) data swapping between two different logging services on different platforms (since it is possible that other systems, like hospitals, will want to communicate and exchange information with the SOCIAL platform). For example, someone that is associated with our project may want to follow the same principles as our *Logging* service and be also able to trade block and log information for redundancy reasons (decentralized database). The architecture could be something as depicted on Figure 3.5, where the **label 1** in the arrows represents the communication and exchange of data between the different logging services (the messages inside may vary, according to each logging service's restrictions) and the **label 2** represents the exchange of information between the logging service and its respective DBMS (to retrieve or store data).

As a side note, one logging service may want to share (or is asked to) its database content from another (or multiple) logging service. However, the other logging services must verify the integrity of this data and follow a process where they can build a message that indicate their trust on that information. Those messages can later be used to prove that our information was validated by someone else, and they can not deny that. This will prove useful when exchanging information with other systems and know which data is validated or not, across different databases.

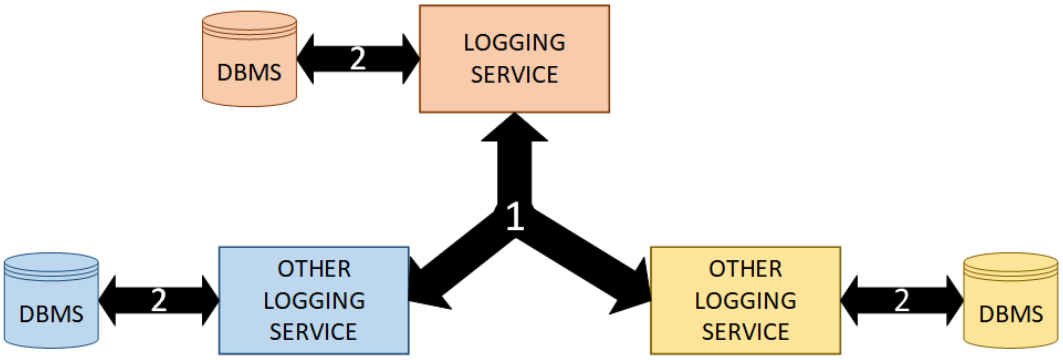


Figure 3.5: Communication between different logging services

3.4.1 Multiple Blockchains

Regarding the blockchains themselves, there should be two kinds of them. One per user (for all of the different services he uses) and one per service (for all of the users that use it), so it can be possible to have more persistence and to be able to construct a timeline for both the users and services and make it easier to detect the exact point of an anomaly. Each of these blockchains will work as an overlay to the logs of each service (see examples in Figure 3.6) or user (see examples in Figure 3.7) and that will allow to confirm that during that time period, those events are consistent and can be trusted. The logs itself are not going to be modified since all of the block's data is the metadata (information of information) for the respective logs.

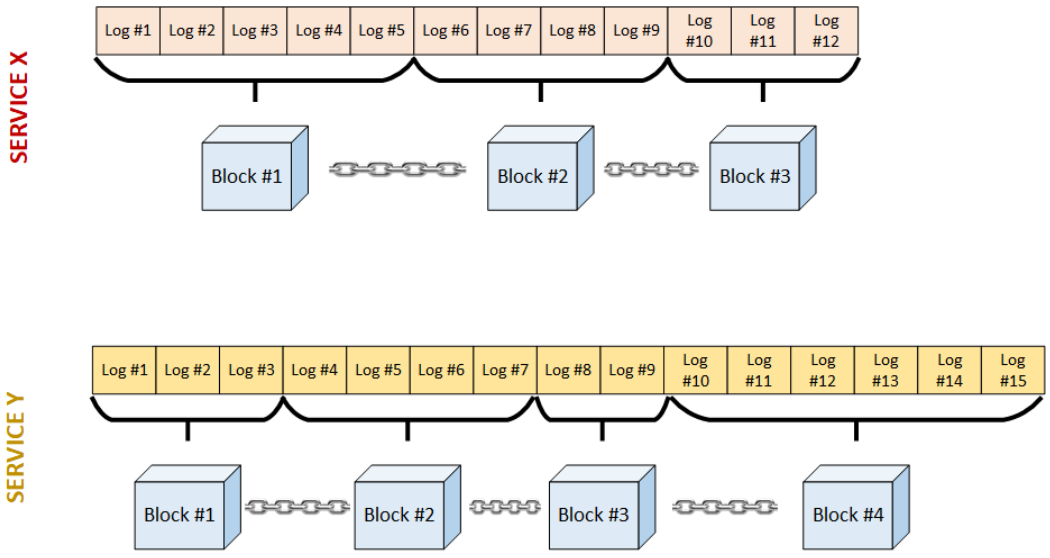


Figure 3.6: Example of two services' blockchain overlay

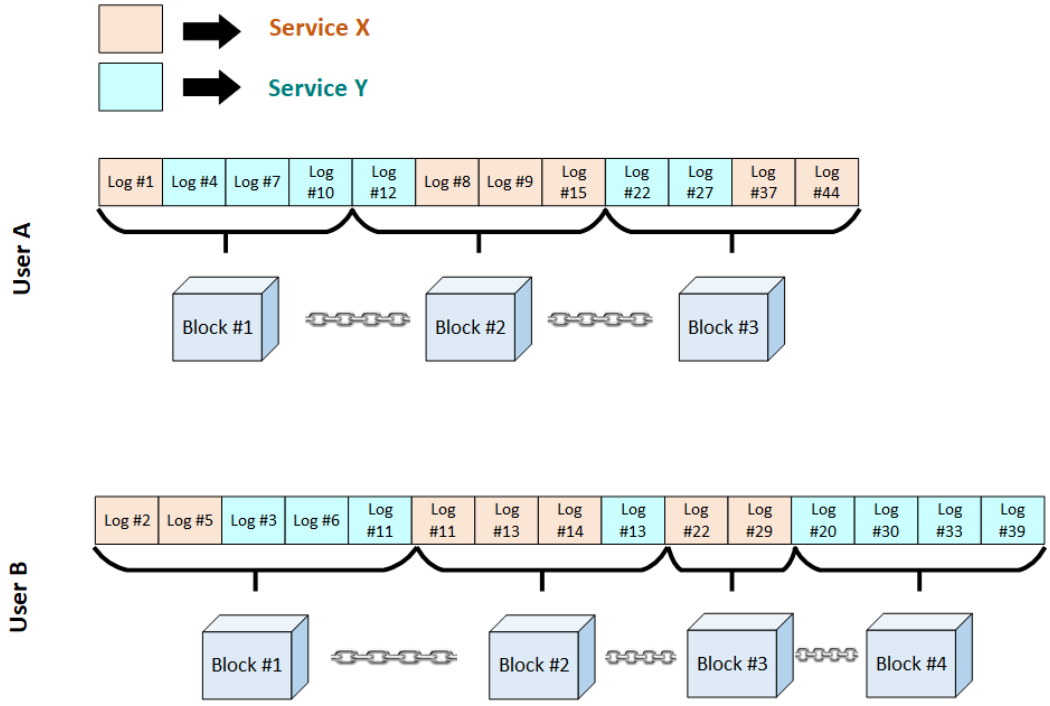


Figure 3.7: Example of two services' blockchain overlay

Figure 3.6 shows that, for each service, we will have a blockchain where every block is associated with a stream of logs (sorted by their timestamp). That means that we can guarantee the integrity of the timeline of events that occur inside the service, if the value of the respective block's hash matches the re-calculation of the hash for the corresponding events. Figure 3.7 follows a similar logic, but now, the stream of logs are not associated to one service, but for multiple services that the user accessed. The timeline of events is constructed by retrieving all of the logs across all of the services corresponding to that user (between two time periods) and sorting them through the respective timestamps. For example, one user's block may have three logs associated, where the first one is the user's successful log-in (from the authentication service), the second one is the retrieval of its role in the system (from the authorization RBAC service) and the third one is the log-out (authentication service again). These logs are sorted by their timestamp, and if we re-calculate the hash of the three events and it is the same of the respective block's hash, then we can guarantee that there was no tampering during the timeline of events of that block.

3.5 Use Cases

By following the architecture presented on both Figure 3.4 and Figure 3.5 and respective requirements previously discussed, several scenarios can be fulfilled. Three possible use cases, for the logging and auditing processes (including the access control), will be demonstrated in

this section.

3.5.1 Storage and Retrieval of Events from a Service

This use case involves one of the services that generates the logs: The authentication service wants to store its log entries for future analysis. It sends every activity that happens whenever someone accesses one of its feature to its logger, the *Logging* service. The *Logging* service store those events inside its DBMS and will create the blocks for that service's blockchain periodically. The authentication service administrator wishes to verify the activity history for maintenance purposes and asks the *Logging* service for all of the events of the current month. After verifying the integrity of the service's blockchain, it will send the requested logs to the authentication service, with the guarantee that nobody tampered with them. The authentication service administrator analyses the results and can conclude it there were malicious events or errors during that month.

Let us assume that it is detected an usual behavior from one user, for example, it was confirmed that the user failed to log-in 8 times, in a short amount of time, before succeeding. Since the service does not want to create a false alarm, it wants to verify the other system's services logs before reporting that user. It requests the logs from that user after the timestamp of the successful log-in, for all services, to the *Logging* service. Before it asks for those logs to the DBMS, it needs to verify if that subject can access those logs for that user for those services. It sends as many requests as needed to the access control, with the appropriate messages and according to the responses received, it will ask the DBMS for the log events for the services that the administrator of the authorization service is allowed to access. It will send all of them to the administrator and now he can verify if that was an attacker trying to manipulate something on the other services or if it was a simple mistake (see flow example of Figure 3.8).

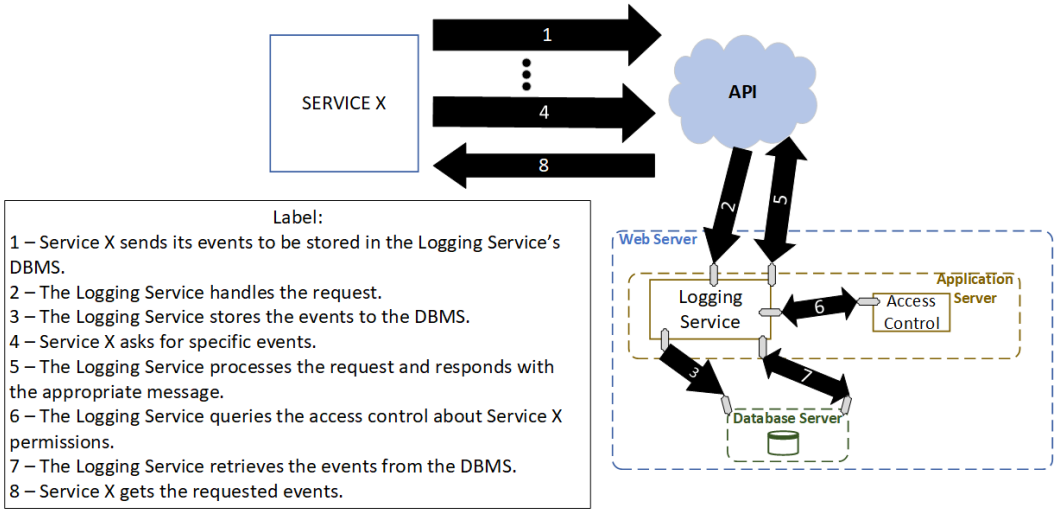


Figure 3.8: Use Case Example 1.

3.5.2 Auditing request from a Doctor

There is somebody who wants to access logs from someone else: One doctor needs to verify if its patient is not having any problems at home. That doctor knows that the patient wakes up every two days at 8 a.m. to clean his house. Every time that he cleans he opens several windows and doors, all of them with a motion sensor attached. The doctor requests to the platform all of the logs from that user (from the sensor detection service). The *Logging* service gets that request and, before requesting those events from the DBMS, first it must send a request to the access control to know if that doctor has permission to access the logs or not. In these situations, the FHIR resources can come in handy. If we know that every family doctor should have access to the logs of its patients, then when we build the policy file, we can add that stipulation and create the respective rule. After verifying that the doctor can see the user's logs from that service, the *Logging* service will request the data from the DBMS and send it to the doctor. The doctor analyses the timeline, to see if for every two days, in the morning, the door and window sensors activated. If he observes that during a long time period none of them activated, that can mean that some accident happened to his patient, inside the house (see flow example of Figure 3.9).

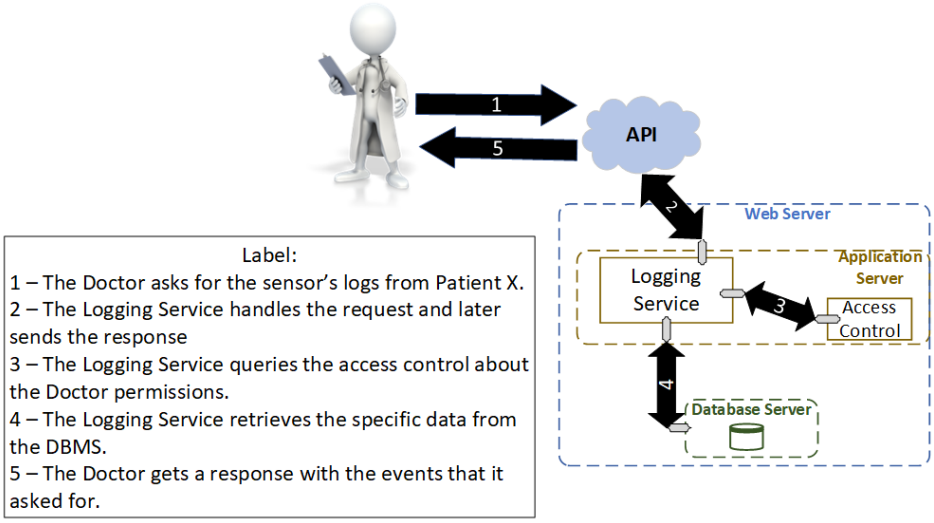


Figure 3.9: Use Case Example 2.

3.5.3 Exchange of data between different logging services

This case explores the interaction between different logging services: Another logging service wants to exchange log and blockchain information, for redundancy purposes. Every time that the block insertion process is concluded, our *Logging* service will send the block and logs associated to the other service. It will analyze the validity and integrity of the data received and send a signed message (to the requester) that proves that everything is verified and stores all on its DBMS. The same can be done vice-versa. There now can exist two sub-scenarios.

First, our *Logging* service needs to prove that the integrity of all of its blocks is verified. If

only its testimony is not enough, it can use all of the signatures that the other logging service sent as proof. Even if the other logging service, for malicious reasons, denies to ever validating those blockchains, those signatures can be used against it and prove that the blockchain is indeed verified.

The other sub-scenario is when the other logging service has all of its data deleted (someone attacked its DBMS) and needs to restore its log entries. Since, it was applied a decentralized log database and its information was distributed with other parties, it can request all of that data to our *Logging* service, including our signatures to its blockchains, to guarantee that they are the same events that it previously exchanged.

In the future, we may also have a new (other) logging service that wants to swap information with our *Logging* service. However, that new service is not authorized to access the same logs as the other one. So, every time that we want to distribute our blockchain and log data, we need to ask the access control what entries can we send to one logging service and the other, so that we do not send information that we should not. After obtaining the responses, we can safely send the appropriate logs and blockchains to the respective logging service, without revealing too much (see flow example of Figure 3.10)

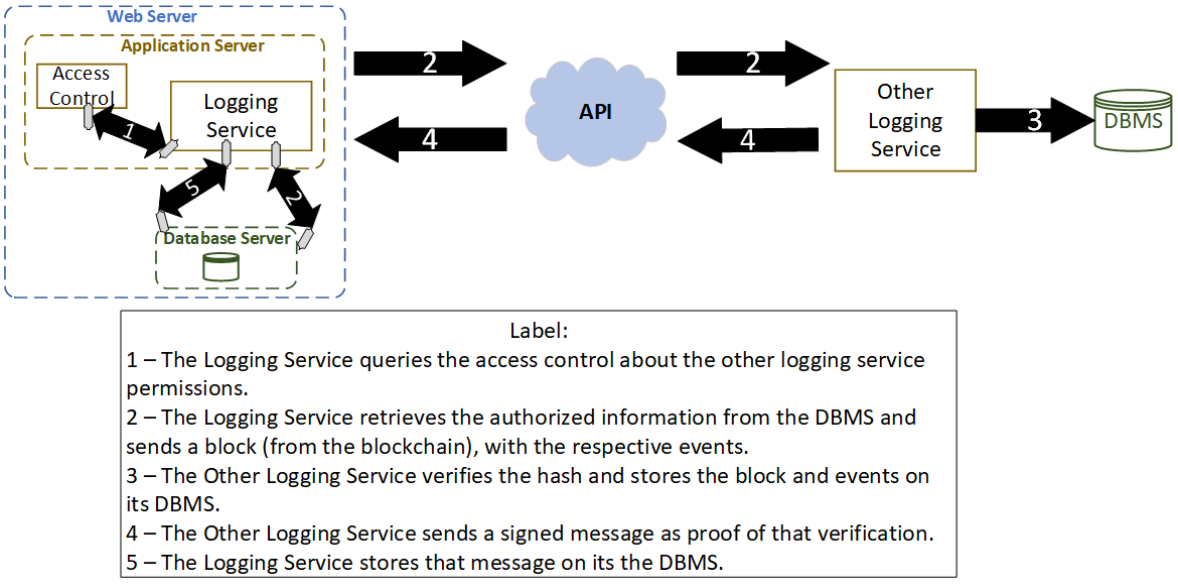


Figure 3.10: Use Case Example 3.

With the requirements, architecture and use cases defined, we can see that integrating the blockchain technology to a logging system not only solves the integrity issue that can occur when there is a great quantity of data, but it can also be implemented in a way that can be flexible to any type of requirements and restrictions issued by other services (whether they are the services inside the same platform or other logging services from other platforms) and have a redundant log storage, since it follows a decentralized database concept. There is a future for this type of technology and it is not only exclusive to cryptocurrency.

Furthermore, considering that we are dealing with logs that can contain Electronic Health Records, relying on a RBAC is not such a great solution because a role should not be the only attribute that dictates whether someone should have access to a specific piece of information of another person or not. Every facility, city, country and continent can have different rules and it is unwise to create such a generic and simplistic method to define those rules.

Overall, the best alternative is having an ABAC implementation and the XACML not only follows this model but also offers the tools to create a set of policies so thorough that even the most strict regulations can be efficiently integrated.

In the following chapter we will explain how it is possible to apply all of these these concepts to a real product and how to develop a *Logging* and *Auditing* service with access control and FHIR integration. Since we do not want to reinvent the wheel, whenever exists a reliable and open-sourced product or framework that can be used for one of the components, such will be indicated and explained why it was used. However, not all of the components can originate from existing APIs, since they would be incompatible to attain our goals (this will also be justified, when discussed further on).

CHAPTER 4

Implementation

On this chapter it will be specified what technologies are used for the implementation of the different components related to the Logging service, that was talked about during chapter 3, including some of their features and advantages.

Following the proposed requirements, architecture and use cases from Chapter 3, it is possible to develop a prototype of the *Logging* service. Knowing what the goals are and using Figure 3.4 and 3.5 as a basis, section 4.1 explains how the *Logging* service is built and what feature it has available to the other services that want to send or exchange log or blockchain data with it and specifies the access control component (that follows the architecture of the XACML standard, Figure 2.15) that is integrated with the logging service and how they communicate among themselves, while section 4.2 reveals what application server is used to deploy that service. Section 4.3 elaborates on the DBMS used to store the log events and blockchain information and its advantages, and section 4.4 details the blockchain implementation and why existent APIs were not used.

4.1 Structure of the Logging and Auditing service

Since we want our *Logging* service to be able to exchange messages with the outside (the consumers of our service) and that it should be the only one to directly communicate with the access control (for the auditing process) and database and also have a timer module that would send warning messages periodically, the general structure of all of those components follow the one from Figure 4.1.

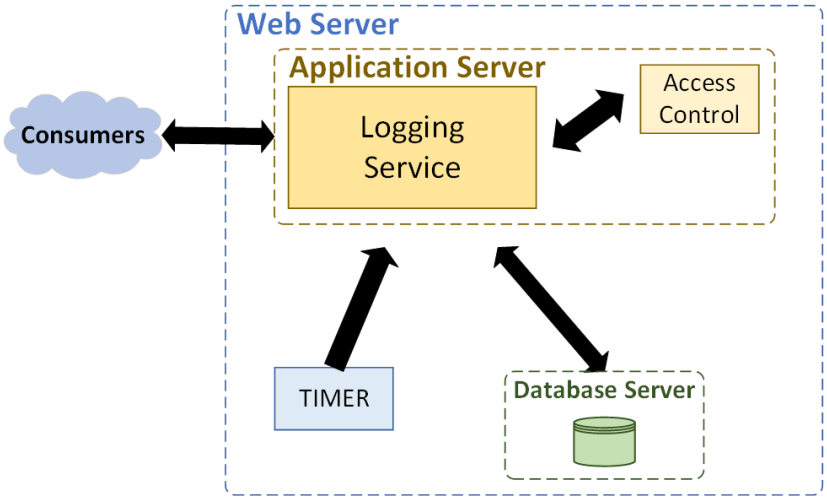


Figure 4.1: *Logging* service architecture

Since the Java language is object-oriented (code can be reused), platform-independent (it can run the same program on different systems) and has many libraries and frameworks available (either by official sources or by the developer community), it was used to program all of the methods of the *Logging* service. Additionally, the messages exchanged between the DBMS (that will be detailed later) and the access control (that uses the XACML standard) can be processed and created using that language, including the security functions (like hashing or signing) that will be used during some operations.

Inside the web server, where all of the components will process requests, we have an application server, where the *Logging* service and the access control will execute the requests received. Because the DBMS has its own server and since the timer component only sends simple warning messages to the *Logging* service (it does not have any feature where it receives and processes a message, it can be a simple program that can be executed in the background as a process and sends requests periodically), they do not need to be deployed inside the same app server.

For a more performance-efficient communication, the different services will use a stateless communication protocol and since our service is supposed to be constantly available and running it makes sense to make it a RESTful Web Service. Using the REST architecture allows us to handle the exchange of several resources at the same time, to expand (the *Logging* service) with ease and to work on the Web, via URIs. In addition to this, we can categorize each of our service's functions into five types of operations: Create (HTTP POST), Read (HTTP GET), Update (HTTP PUT or HTTP PATCH) and Delete (HTTP DELETE). Also, if we integrate the Swagger [111] tool with our RESTful Web Service, we can create an interactive documentation website that can help the other services know how the communication works, via a User Interface, for every request that our service can handle (it can have a description, what are the parameters, what are the possible responses, ...).

For the authentication of the messages sent between the different services we can either

add a signature field and value so that the services can validate them or by using one of the other methods presented on chapter 1 (section 2.2). The OAuth will be the ideal method to use and the logging service will only accept messages that have the correct access token.

If a digital signature is the more viable option, then there needs to be some sort of key management, because it is necessary to establish a secure way to exchange the keys that are going to be used and what method is going to be used to forward a process that requires multiple validations from different modules (e.g.: the construction of a block to the blockchain).

Next, we are going to discuss the different features of the *Logging* service. From the outsider perspective, it is possible to Request Log Events between two time periods or identifiers (specified by the requester). Whenever the *Logging* service gets one of these type of requests, it analyses if the requester has the permissions to access that data and if it does, retrieves the information asked from the DBMS and sends it to the consumer (Figure 4.2).

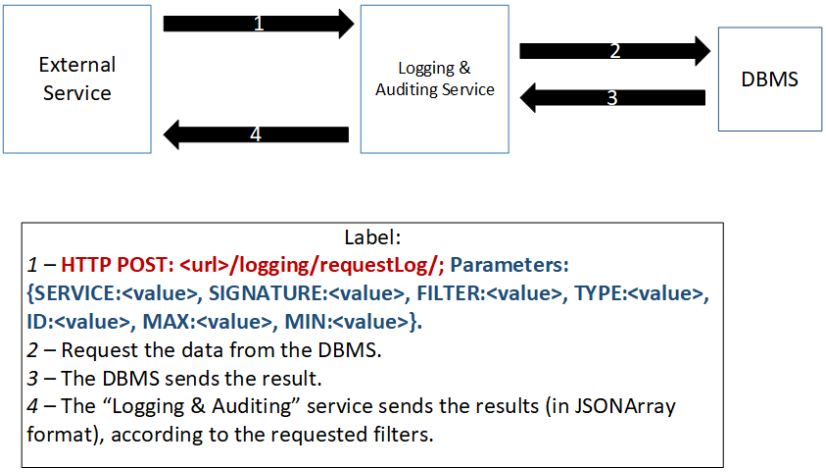


Figure 4.2: Request Log Events

For the block insertion process, the *Logging* service and the other service may have different roles, depending on the type of relation established beforehand.

The most simple interaction is when the other service does not need to approve the insertion of the new block to its blockchain. The *Logging* service is fully responsible for all of those operations. Figure 4.3 shows this type of interaction, where after receiving the message from the timer, indicating that it should start a new block insertion process, the *Logging* service creates and stores the new block for the respective blockchain directly.

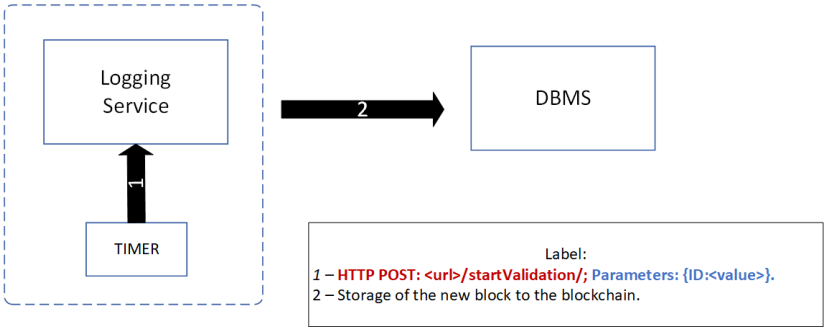


Figure 4.3: Block Insertion (Option 1)

Another feature is allowing the other service to approve the insertion of the new block before the *Logging* service adds it to the DBMS. The *Logging* service will send the an array of the events that is about to hash and if the External Service approves, it will hash them and add it to the block and insert it to the respective blockchain (Figure 4.4).

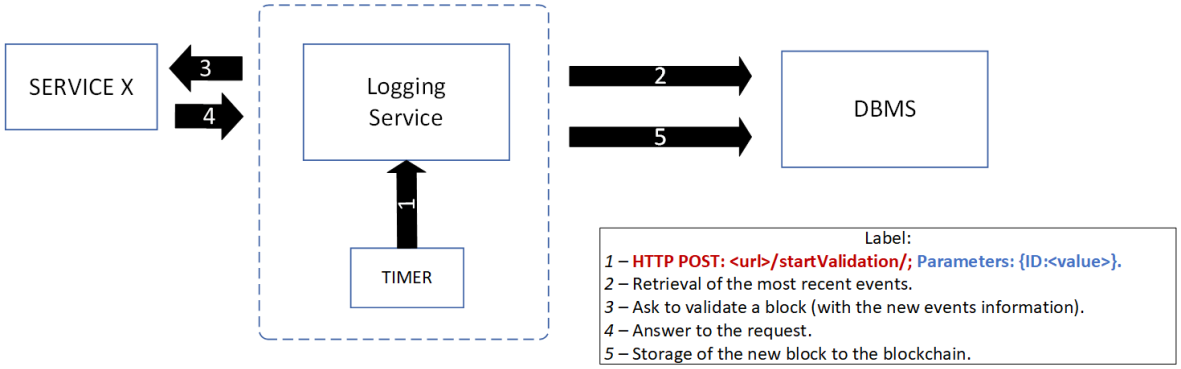


Figure 4.4: Block Insertion (Option 2)

A third alternative is, instead of the other service sending the events on its raw format, it will send an hash value of that event and when it is time to insert a new block, the *Logging* service will send the array of the most recent hashes received to the External Service. If it verifies the consistency of those hashes, it will send the raw data of the events, so the *Logging* Service can store them on the DBMS (example, Figure 4.5).

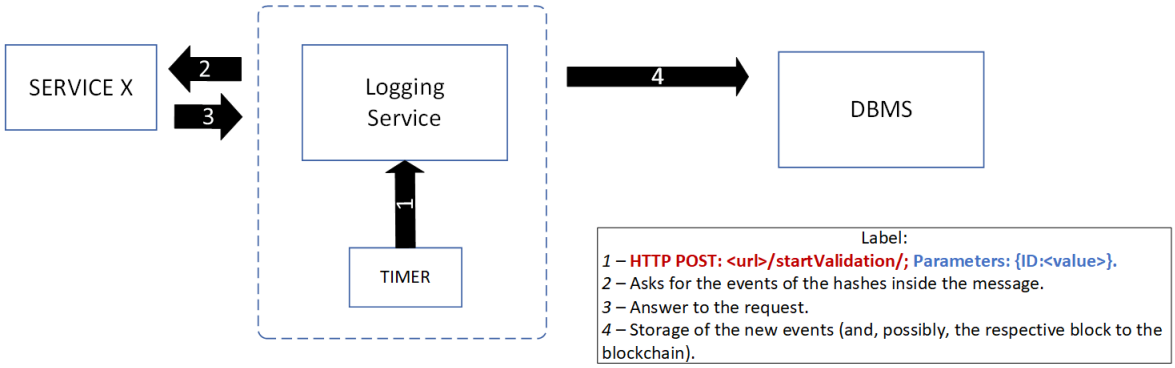


Figure 4.5: Block Insertion (Option 3)

The final option is the scenario where the other service wants to be the one to hash the events. So, the *Logging* service will send the array of the respective events, and the External Service will hash them and send the result to the *Logging* service, so it can add that value to the blockchain (example, Figure 4.6).

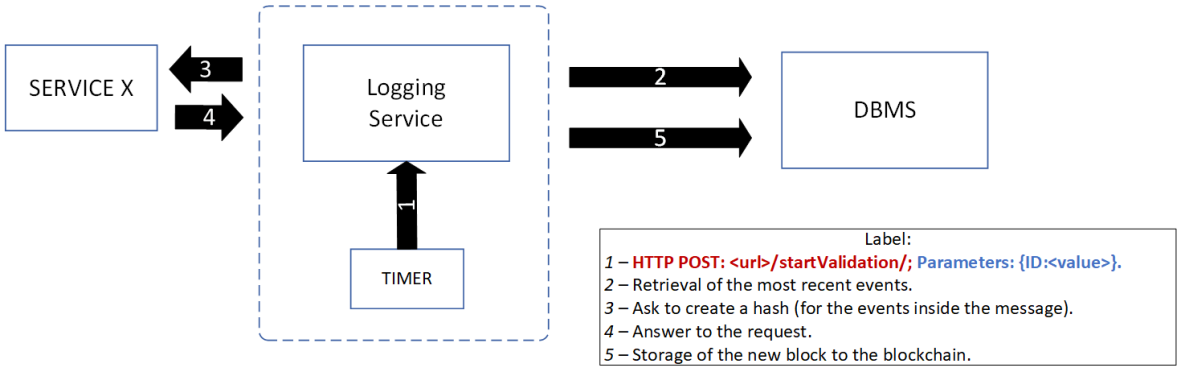


Figure 4.6: Block Insertion (Option 4)

For the messages exchanged between different logging services (Figure 3.5, chapter 3) that will store the log events and/or blockchains of different systems (distributed database), the flow of these exchanges is represented in Figure 4.7

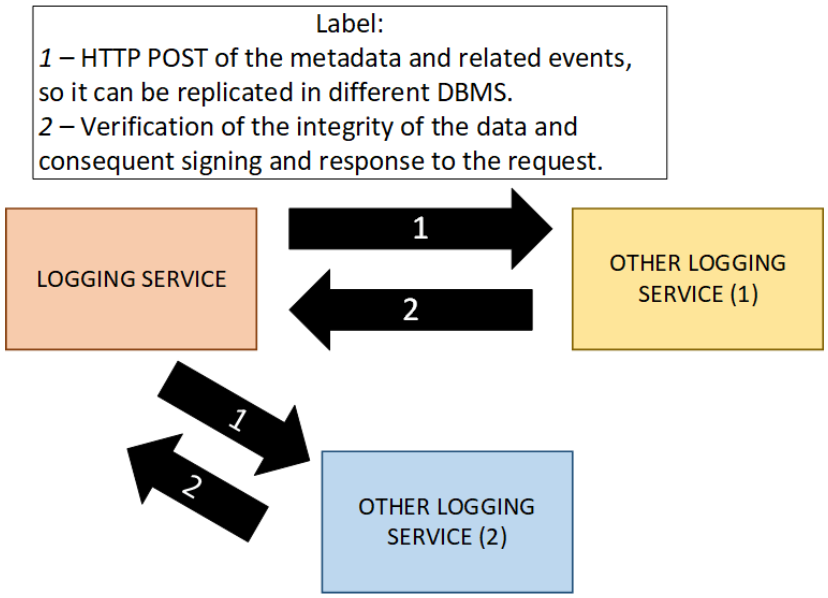


Figure 4.7: *Logging*' data exchange flow

As we can see in Figure 4.7, initially the *Logging* Service will send an HTTP POST to one or more other services and if the other services verify the legitimacy of that block they will respond with a message with that verification.

For the verification of the blocks from external logging services, they will follow the trusted timestamping procedure, explained on chapter 2, section 2.3 and send that value to the

requester service so it can store it and prove in the future that that service verified that block at that time instant.

For the access control component, as mentioned during the document, the XACML standard is used. Of all of the different entities of the XACML architecture, the PEP is the one that is developed by us, because how the decisions of the PDP are enforced vary according to each system's goals and objectives. Some PEPs may interpret a *Deny*, *Permit* or other possible attributes in different ways than others. For our purposes, our PEP will be a translator of the requests sent by the *Logging* service to the access control. It will receive a JSON message with the `subject` (service or user who wants to access), `action` ("read"), `resources` (list of the users that the subject wants to access) and `environment` (list of the services that the subject wants to access), transform it to the appropriate format and send it to the PDP and waits for the response. It will send that response (*Permit* or *Deny*) to the *Logging* service and then it will know how many of the log events it can show to that subject.

Since the PDP's main feature is to compare the attributes and respective values of the request received from the PEP to the ones inside the policy files, it is possible to use an already developed one and integrate with our ABAC.

In our case, the AuthzForce framework [28] will be used, since it uses the XACML standard (OASIS XACML standard v3.0) and it provides a HTTP and REST API (for the Policy Decision Point and Policy Administration Point).

Not only does this free API manages to do efficiently the main objective of the PDP (compare two attributes and its values and give a *Permit* or *Deny* response) but it also has integrated several of the features (mandatory and optional) of the OASIS standard.

With these features we can send more accurate requests and have more detailed decisions. This PDP can interpret over ten different data-types, including `date` (e.g.: 2018-12-31), `ipAddress` (e.g.: 172.16.254.1) and `xpathExpression` (e.g.: city/bookstore/shelf/book) and over two hundred different functions (for example, if a value is less, equal or greater than another or if an array of values of an attribute contains a value of a request).

Plus, the PDP can now send a response message with more fields than the `Decision` one (that contains the values *Permit*, *Deny*, *NotApplicable* and *Indeterminate*). If needed, it is possible to add a `Status` field where the value will be the `StatusCode` (*ok*, *missing-attribute* or *syntax-error*) or even a `Advice` field for supplemental information (for example, a request to access the logs from the "Authentication" service and depending on the subject, the PDP would send a *Permit* where the `Advice` value would be the days of the week that that subject could access). This allows the PEP to perform different actions even if it receives the same *Permit* or *Deny* result. More features and details can be seen on the official AuthzForce's PDP features page [29].

If a FHIR integration is required then the FHIR server has several resources available and the ones that should be used to construct a policy document can vary. But for our case, we

need the ones that tell us what are the relations between two entities and the entity's data and information itself. So, we will focus our attention on the following resources: *Patient*, where the most relevant data is the `ID` (identification of the *Patient* on the FHIR server), `IDENTIFIER` (an array of the different identifications that the *Patient* has on several systems), `NAME` (Family and Given), `ACTIVE` (true or false) and `DECEASED BOOLEAN` (true or false); *Practitioner*, that has a data structure to the *Patient* resource, like `ID` (identification of the *Practitioner* on the FHIR server), `IDENTIFIER` (an array of the different identifications that the *Practitioner* has on several systems), `NAME` (Family and Given) and `ACTIVE` (true or false); *Organization*, the most significant data being its `ID` (identification of the *Organization* on the FHIR server), `IDENTIFIER` (an array of the different identifications that the *Organization* has on several systems), `NAME` and `ACTIVE` (true or false); *RelatedPerson*, that has as attributes its `ID` (identification of the *RelatedPerson* on the FHIR server), `IDENTIFIER` (an array of the different identifications that the *RelatedPerson* has on several systems), `NAME` (Family and Given), `ACTIVE` (true or false), `PATIENT` (JSONObject with the identification of the person involved with this *RelatedPerson*); *CareTeam*, with `ID` (identification of the *CareTeam* on the FHIR server), `SUBJECT` (Identification of the *Patient*) and `PARTICIPANT` (Array of the different participants of the *CareTeam*) as the more pertinent data. All of these resources have also other data that can be used for additional information and/or more restrictive policies, like `Address`, `Telecom`, `Gender` or `BirthDate`.

The *CareTeam* resource shows what kind of relation exists between the *Patient* and the *Practitioner*/*Organization*/*RelatedPerson*, more specifically, their roles (e.g.: "Employer", "Medical Expert", "Caregiver", ...). Then, more information can be retrieved from the different resources including the time limit of their role on a certain subject/resource. Using all of that data, the policies can be constructed and we can know what kind of actions certain subjects (*Practitioner*/*Organization*/*RelatedPerson*) can do to specific resources (*Patient*). For example, we can associate an action (for example, "read") and a type of service ("authentication", "authorization", ...) to a subject's role (if it is a "Medical Expert", then it can "read" the logs from the services "authentication" and "authorization" of its *Patients*).

Figure 4.8 shows an example of a XACML message type (that was adapted from a simple JSON request) that is going to be interpreted by the PDP.

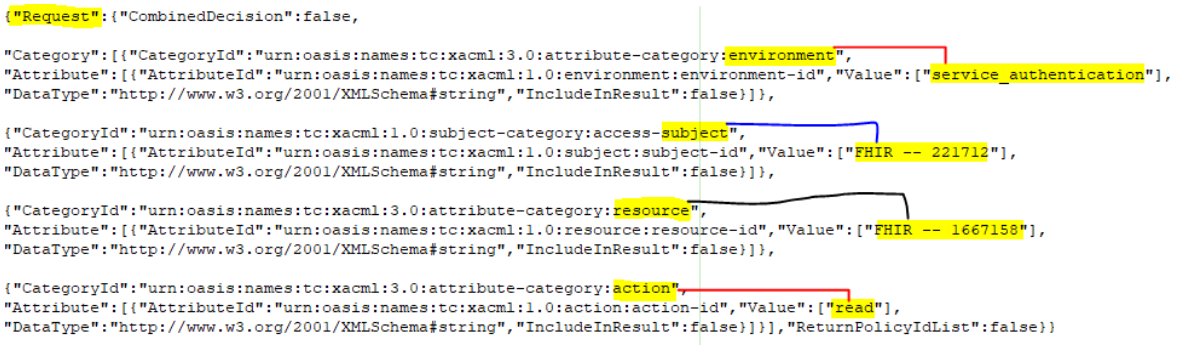


Figure 4.8: XACML message type.

With the example presented (4.8) there is a request from a `subject` to read (`action`) the log entries of that person (`resources`) from that service (`environment`). If the PDP has a rule that permits those specific attributes with those values, it will send a *Permit* response and the *Logging* service will send the data requested to the requester. If not, it will send a *Deny* response and the *Logging* service will not show the data requested to that `subject`.

How our Access Control is developed is so it can decide the access for the services that send the events, the other logging services and users that are identified on FHIR. With this scheme we can adapt our system for different types of auditing. We allow services that want to request their own logs or from other services, so that they either store them on their own DBMS or to construct the timeline of events of an user across the different services. The other logging services can also make requests for logs or blocks data. We also permit medical experts or other people related to a user/patient to be able to view their timeline and find out what were their actions during a certain time period. Only if these subjects have the right permissions can they access that data and those requests will also be stored as log events for evidence for possible attempts of misuse of information.

4.2 Application Server

Since the *Logging* service needs to exchange data with other services, either they be internal or external to the project's platform and needs to be available during long periods of time, we are developing it as a Web Service (more specifically a RESTful Web Service). After creating the web service, we need a environment so that it can run and to make it accessible to whoever wants to communicate with it. We will need an application server.

The server that is going to be used will be Apache Tomcat [5], considering that is one of the most recommended to use for any web developer to deploy their services. Not only is open sourced but it is also very lightweight, very simple to use and implements several technologies that many applications normally use.

After installing it on a machine we then need to create the artifact of our service (for example, a Web Application Archive (WAR) module) and move it to respective Tomcat directory (in our case <tomcat_rootdirectory>/webapps/ROOT/). If the Tomcat server is running, then the web service will be deployed and ready to be interacted with.

As it was mentioned in section 4.1, the PDP used for the Access Control will be the AuthzForce RESTful PDP [30], that is packaged as a Spring Boot application [91]. Since Spring Boot applications do not (by default) need to be installed or to set up servlet containers to be deployed, to be able to deploy on the Tomcat server some modifications need to be done to certain files, so that it can create a WAR file to later be deployed into Tomcat (a simple guide can be found on the following *Mkyong.com* page, "Spring Boot – Deploy WAR file to Tomcat" [74]).

Tomcat can handle several WAR files at the same time, but first we need to setup a connector for each port (that will be associated with a WAR file) and move them to differently named directories inside the Tomcat's "/webapps" directory. If all of this is one done correctly, we can have all of the logging, auditing and access control features inside the same application server, where each interface has its respective address port that can be accessed.

4.3 Database Management System

Data needs to be stored and retrieved quickly and efficiently, so that we can access specific information in a short amount of time and without compromising their consistency inside the system. One information may depend on other and if there are corrupted or lost data, the entire system's integrity may be lost forever.

Which database management system to choose depends on which factors are most important for the project in question. If logging is the main focus, then a search engine is ideal. Search engines are very useful when the main focus is searching for data content and it can also deliver precise search results (through detailed filtering) at very reduced time frames. In our case, we will be using ElasticSearch [11].

ElasticSearch is a document-oriented model, more specifically JSON documents. It can record any change made in the system, to prevent data loss and it can be used for full-text searching (search any data for the best match with a specific phrase). Since it uses indices, they can be divided into shards, and it is possible to replicate them. It is also open-sourced and highly scalable. This search engine is used by Stack Overflow, Medium, Tripwire and Netflix [108].

The data can be stored (in ElasticSearch) in a specific index. First, we need to create the index, and then we can associate a mapping to it to indicate what fields and data type the ElasticSearch will accept (see example on Figure 4.9). If we send a JSON object to a certain index and it follow its mapping structure, it will be successfully stored.

```
PUT my_index
{
  "mappings": {
    "doc": {
      "properties": {
        "title":    { "type": "text"  },
        "name":     { "type": "text"  },
        "age":      { "type": "integer" },
        "created":  {
          "type":   "date",
          "format": "strict_date_optional_time||epoch_millis"
        }
      }
    }
  }
}
```

Figure 4.9: Example of an index (ElasticSearch) [32].

The data types that ElasticSearch supports include string, numerical (long, integer, float, ...), date, boolean, binary and object (JSON). If an ID is not specified when inserting new data, a random value will be generated for that field.

By using this JSON format, searching becomes a very efficient process since we can have more detailed queries. We can ask for every result of an index that as certain value in a specific field and ElasticSearch can retrieve those results in a quick manner.

ElasticSearch has several plug-ins and can also integrate several other products that can help with security, management and parsing. By using Logstash [34], it is possible to parse the events that are sent and retrieve the appropriate information needed to store into ElasticSearch. Kibana [33] allows for different types of visualization (through histograms, line graphs, pie charts, ...) of the data that is store in ElasticSearch and also allows, with X-Pack [35], to integrate User Authentication whenever someone wants to access this DBMS. With all of these features integrated into our DBMS, we can already guarantee some security to our stored information, a good performance for the different data operations available and the usage of several data management techniques, making the choice of using ElasticSearch as our DBMS a perfect fit for our goals and needs.

4.4 BlockChain

Considering that there are two types of blockchains that are going to be used and we want to have a more efficient way for preventing tampering with our data, we are not using any existent API of blockchains (for example, the most commonly used, the *BLOCKCHAIN* API [112]), but instead we are creating our own. By using the good practices of the established existent API's, we can create a blockchain structure that is more efficient for our goals and objectives.

To better understand why we created a new and different blockchain, let us compare it to the one mentioned earlier [112]. It works through the exchange of JSON messages and the block data itself is also on the JSON format. However, just like with most other blockchain's API, its structure revolves around cryptocurrencies (example, it offers the possibility to subscribe to be notified whenever a new block appears or transactions for a certain cryptocurrency address). Because of this, the blockchain metadata has certain information that is unnecessary for our logging and auditing purposes. For example, the API has a field (`tx`) where the value is the array of transactions. Other API' fields are unnecessary for the project, like the `size` (of the block) and `nonce` (a value used in the hash calculation, so that the result can have a certain amount of leading zeros). Other APIs like Coinbase [6], Bitcore [14] and Colored Coins [24] follow similar infrastructures, where they are developed with a cryptocurrency scenario in mind.

Since we do not need that much information and we do not want to have every data of that block available in the open, we decided to use the following fields:

- `Hash`: the hash of the current block (encoded, using the Base64 scheme, for interoperability purposes), calculated using the *SHA-256* cryptographic hash function and the concatenation of the content of the stream of events and the previous block hash (if existent) as the original value.
- `nextHash`: the hash of the next block in the chain, NULL if not existent (encoded, using the Base64 scheme).
- `prevHash`: the hash of the previous block in the chain, NULL if not existent (encoded, using the Base64 scheme).
- `maxCheckpoint`: the identification of the last log event of the set.
- `minCheckpoint`: the identification of the first log event of the set.
- `minTimestamp`: the lowest timestamp of the log set (in milliseconds).
- `maxTimestamp`: the highest timestamp of the log set (in milliseconds).
- `Identifier`: the identification of that block on ElasticSearch.

Each of the log events, stored in ElasticSearch has the following fields:

- `Content`: the log's data that was sent by the service (example, "UserA successfully logged in").
- `Event`: the numerical order of the event when it was received by the *Logging* service.
- `TimeStamp`: the timestamp of when that event was stored (in milliseconds).
- `Identifier`: the identification of that log on ElasticSearch.
- `Requester`: who is the subject linked to that log.

Even though we are using these fields, they can extensible to other scenarios. With a more developed system, more fields should be added for more detailed logs and also to add filter options for the search engine.

Every block represents a stream of events and they are linked to other blocks, which allows to prove the integrity for multiple streams, across those respective blocks. That integrity is verified by the hash field of a block. The hash value for the blocks is the concatenation of the all of the log events (between two identifiers) sorted by their `Event` value (descending order). Each log that is concatenated is represented by its `Identifier`, `Content` and `TimeStamp` (the long value is transformed into the respective date format: "DD/MM/YY, hh:mm:ss"), where this values are separated by a comma. For example, one of the log events that are going to be concatenated could be depicted like this: "_nqsq2MBtJsbv0eAbqTz, UserA successfully logged in, 29/05/2018, 12:33:14". Additionally, if we also concatenate the hash value of the previous block (for example, at the end of the stream of logs) we make the block's hash value dependable of the previous block hash and if an attacker manipulates the hash value of one block, he will need to manipulate the value of all of the following block's hashes to hide its tracks (making the tampering detection almost instant). Since ElasticSearch (our DBMS) uses indices, for the log events, each service will have its own index, where their events are going to be stored. For the blockchains, each service and user will have their own index, for the storage of the respective blocks. This structure can be altered to appropriate different systems, but for our purposes, this is more than enough. The other *Logging* services need to know how to interpret these values for both storing purposes and hash calculations.

Like it was explained before (chapter 3, section 3.4) having two types of blockchains, one for each user (and its stream of events across different services) and for each service (and its stream of events among the different users) can help to verify the integrity of the timeline of different flux of events. Because of this, we need a specific structure for the blockchain that will allow for to efficiently query the DBMS for certain data and according to our own goals and objectives (that are different from the existent APIs, that focus more on cryptocurrencies, where we instead focus more on logs). Using the `min` and `max` fields make more sense than the array for all of the transactions of the API, because for our logging and auditing purposes, we only need to know, to construct timelines from the stream of events, the first and last event and not all of the log events associated with that block. For example, by using the `maxTimeStamp` and `minTimeStamp` fields in our blocks, we can later use that information for searching the events that are associated with that block for detection of possible tampering or just to retrieve some data for other operations.

Probably the biggest obstacle when using existent blockchain' APIs is their usage rules and restrictions, and usually that involves making our distributed database public, outside of trusted parties. Since we have a *Logging* service that will communicate with several other services and each one may want to create restrictions and rules regarding the access from other parties over their data, if we create our own blockchain we can also implement an access control service with it and establish our own rules of access.

Because of the simplicity of the blockchain's structure, almost any developer can easily create their own one and modify it to serve their work needs. Using existing ones may lead to modifications of some features of the project so it can be possible to integrate it. So, in the long run, it is better to develop a customized blockchain that fits the goals that we want to accomplish.

CHAPTER 5

Evaluation and Tests

With the introductions (Chapter 2) and development explanations (Chapter 3 and 4) out of the way, all of the concepts that were previously talked about, need to be tested to prove that they are in fact the best alternative to solve the problems caused by other proposed solutions. This chapter will present different scenarios and the respective results and then a conclusion determining if these are the steps that should be followed, when developing the security layer of a piece of software.

To create, to some extent, a realistic scenario and to prove that the implementation could be used on a real product, the architecture of the SOCIAL platform was used as a basis. After analyzing it, it was defined that five services would send their log events to the *Logging* service and that there would be 10 active users for every time frame between new block insertions.

These users will perform different actions over each service at random time instants. When a service performs an activity in the name of a subject it will send a JSON object with the event and the name of the user that is associated (along with other information, if necessary).

The *Logging* service will store these messages in the respective ElasticSearch's index (this facilitates the retrieval of data when necessary and permits a well done categorization of the log events) and every time it receives a request to start the insertion of new blocks to the blockchain it will follow the procedures described in chapter 3. All of the messages are sent via HTTP, since using HTTPS means encrypting the information that is going to be sent, and that increases the workload of the machine and can delay several transmissions (we want the receive and store the several logs from the different services as quickly as possible).

In some tests, because of the limitations of the machines that were used to deploy the different services (the local machine used, a personal computer, had several years of use and the memory and CPU started to become affected by its overuse), some operations would take too long and other operations that depended on those results would use outdated values. So, a minimum time frame between those operations was established during those tests. However, this could prove useful to know how the implementation of the proposed solution handles

with limited hardware and understand better the performance limitations (if a more powerful machine was used, one could be using more than needed without knowing it).

The tests will be divided in two sections. First, we will evaluate the blockchain insertion efficiency for several concurrent services and users, and verify that after some time, the integrity remains consistent. Then, we will evaluate the access control efficiency and certify that only those that have the right permissions can access specific resources under certain conditions.

5.1 Logging and Data integrity

5.1.1 Goals

With these tests, we want to verify that using blockchain is an efficient way to guarantee consistency when dealing with vast amounts of data. We are expecting that after several blocks are added to the blockchain, that every one of them can be analyzed and verified for integrity and that those results come out as truthful. We also want to see if it is possible to create several blockchains, one for each service and each user, without getting mixed up or ending with wrong values.

5.1.2 Test Parameters

The following variables were used for the first test (5.1.3):

- *Number of services*: 5 (Authorization, Authentication ABAC & RBAC, FHIR Gateway and Sensors detection)
- *Users on the system*: 9 (and also an additional "unknown" user, when there is not someone connected to the event)
- *Time between last block insertion and beginning of the new block validation process*: 60 seconds
- *Time between beginning of validation process and beginning of insertion process*: 30 seconds
- *Time between sending events per service*: 1 to 4 seconds (random)

The time frames used were determined through a "pre-testing" phase, where when lower values were used, the blocks would have incorrect values (NULL), because when the DBMS insertion operation started, the hash was still being calculated in earlier operations. Since the machine was very slow during the block creation process, those time frames needed to be established and the ones chosen are the minimum values that could be used (if higher values were used, there would be no problems). The number of users used was also due to similar reasons (when 50 concurrent users were used for testing, the *Logging* service would take too much time to retrieve the necessary data from the DBMS and to calculate the hashes) and to avoid long waiting times (because of the slow processing power of the machine used) 10 concurrent users were seen as ideal for initial and quick tests. If establishing a minimum

time frame between operations is not a viable solution for some systems (the number of users and services in a system can be unpredictable in some cases and defined time frames may not correspond to what is needed), the other alternative would be to create a more complex *Timer* component, that could also handle requests. Every time it would send an alert to the *Logging* service, it would also wait for a response, indicating that the *Logging* service finished the requested process and only after getting this response, the *Timer* would send the next alert to the *Logging* service. But since our minimum time frame option is sufficient for our test parameters, that will not be needed.

The *Logging* service was on a remote machine (*Standard PC (i440FX + PIIX, 1996) 64 bits @ 2GHz, 4GB RAM, 32GB Hard Drive*) and it was responsible for all of the blockchain operations (it did not need the validation or other actions from any service), while the other services were running on a local machine (*Samsung Laptop, Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 8GB RAM, 1TB Hard Drive, Windows 10 Home 64 bits*). Figure 5.1 depicts this structure.

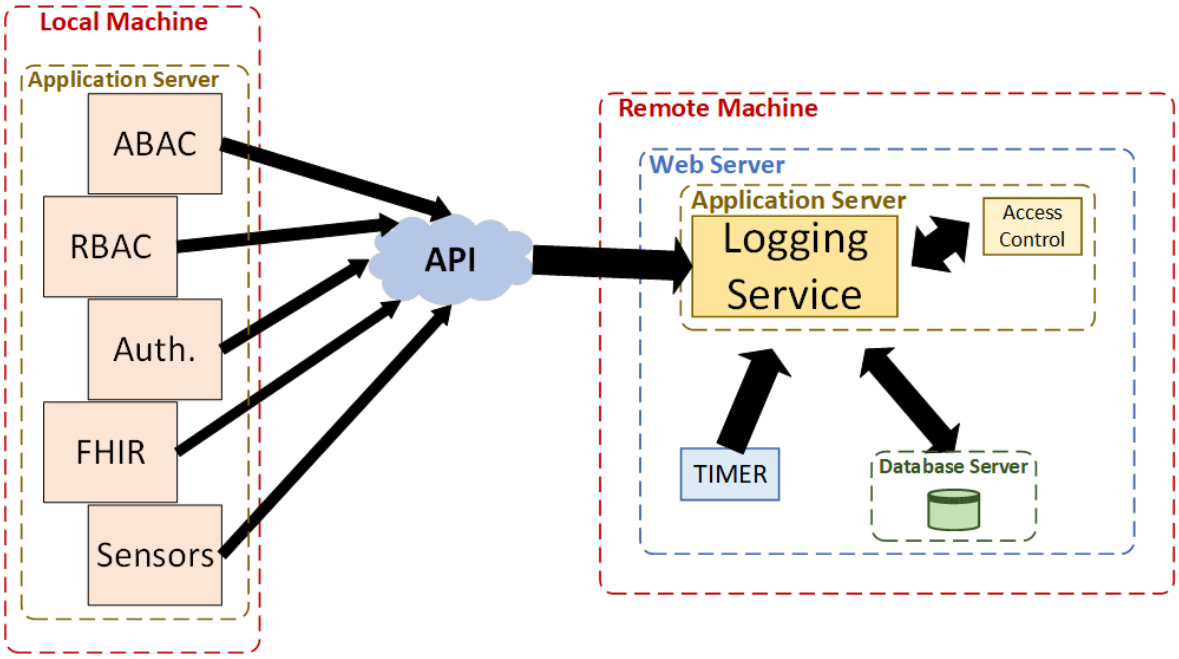


Figure 5.1: Diagram of the architecture used in the tests described in Section 5.1.3.

The second test (5.1.4) had the same variables as the first one. However, all of the services (*Logging* and others) were running on a local machine (*Samsung Laptop, Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 8GB RAM, 1TB Hard Drive, Windows 10 Home 64 bits*) and there were some specifications for each service:

- *Sensors*: Wanted to create the hashes for the block.
- *Authorization (RBAC)*: Allowed the *Logging* service to handle everything.
- *Authorization (ABAC)*: Wanted to be notified and confirm the insertion of the new block.

- *Authentication*: Sends hashes of the events first and then, after validating a certain amount of hashes, it would send the events on their raw format.
- *FHIR Gateway*: Allowed the *Logging* service to handle everything.

Additionally, there were two logging services, that also followed the blockchain method and whenever the original *Logging* service wanted to add a new block to its blockchain, it would send certain data for validation to those services (the first external logging service would receive the block and events data while the second one would only receive the block's data). If validated, every logging service would store the information on their respective DBMS. Figure 5.2 depicts this structure.

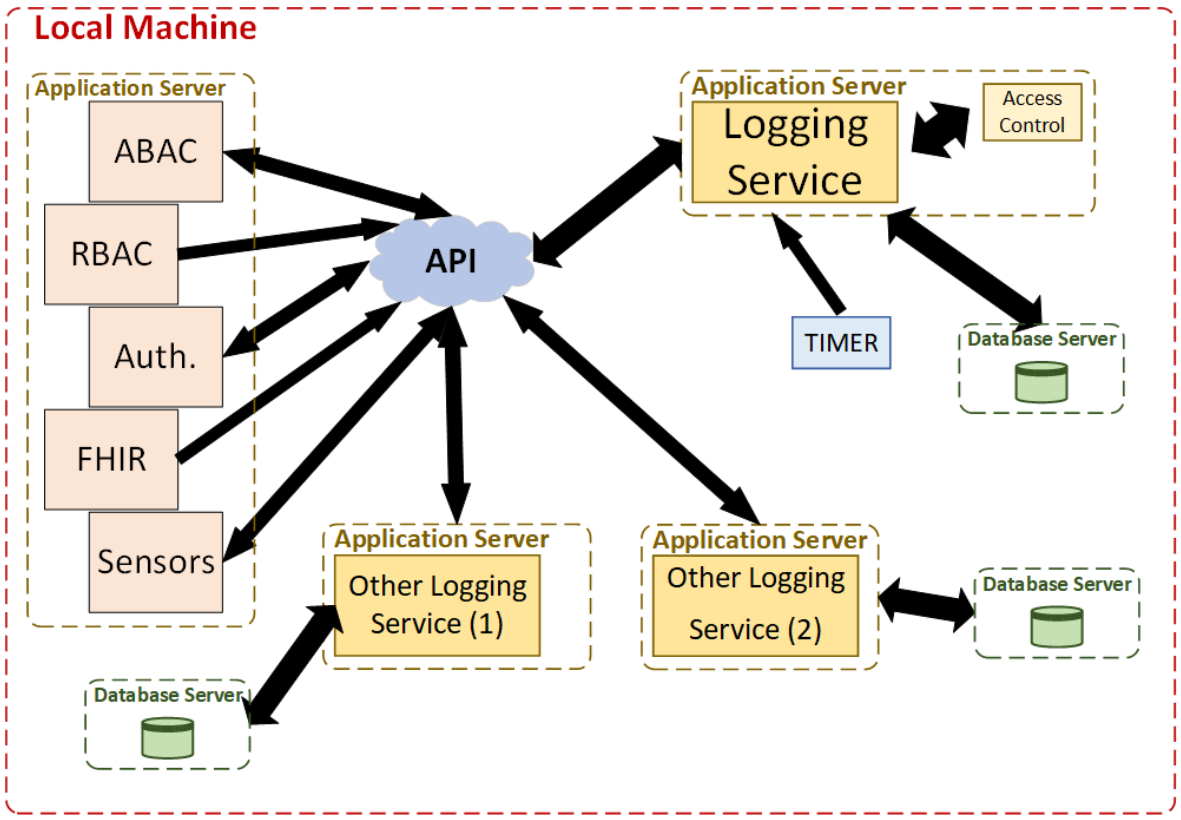


Figure 5.2: Diagram of the architecture used in the tests described in Section 5.1.4.

Figure 5.3 shows how the messages are exchanged inside the system, for the logging process. The services will send, continuously, log messages (with the event and user associated with it) to the *Logging* service, so it can store them in the DBMS. Then, at certain time periods, the *Timer* will send an alert to the *Logging* service, so it can ask the DBMS for the most recent events stored (the *Logging* service knows what was the last event used for the previous block, through a variable used inside the program and then it asks the DBMS for all of the events, sorted by their timestamp, after that value) and send validation requests (for the respective events) to the services who want to be notified and validate the events, before the block insertion process starts. For every validated response, the *Logging* service will calculate the hash and create the service's block associated with the validated events (if a service allowed the *Logging* service to handle everything without needing to ask for validation, it will also

calculate the hash and create the respective block). After a brief period time, the *Timer* will send a alert for the insertion process and the *Logging* service will first ask for the Trusted Timestamping from the other logging services, for each new block it created. Then, it will store the new blocks and trusted timestamping that it received from the other logging services into the DBMS. The *Timer* will send new alerts after a time period has passed.

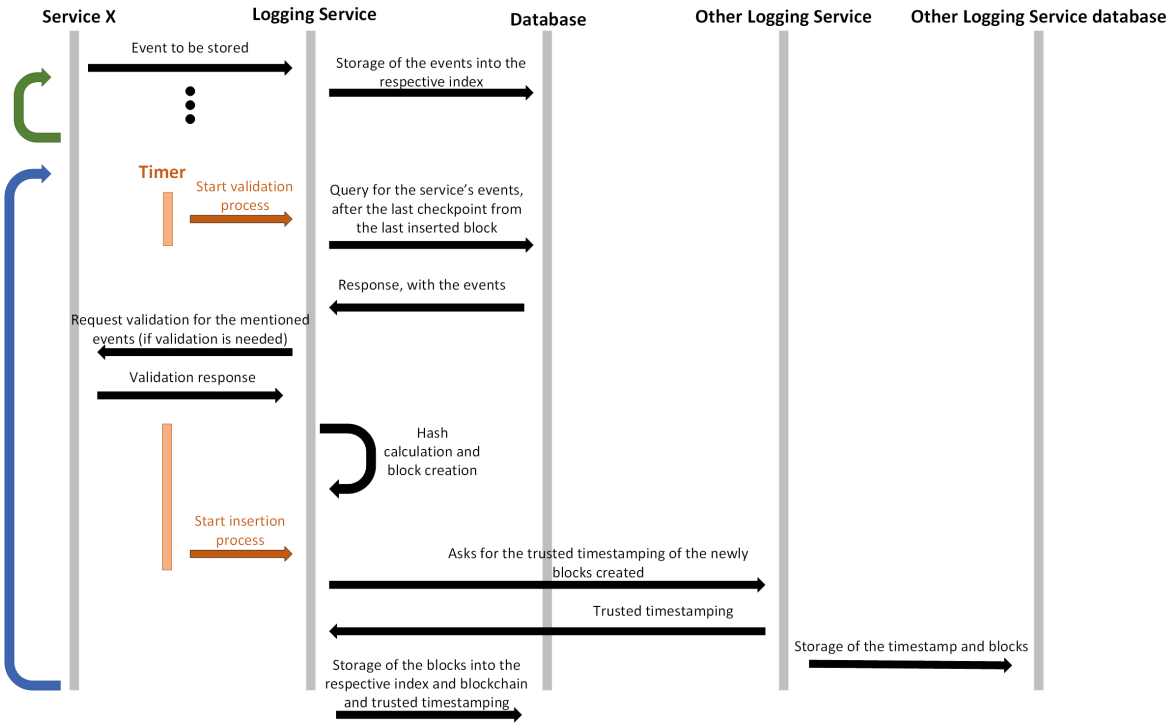


Figure 5.3: Flux of events for section 5.1

5.1.3 Standard Logging Test

This scenario was tested for 20 minutes and all of the services were concurrently sending their events. According to the Table 5.1 results, in average, each service sent 20 messages per minute and in those 20 minutes the *Logging* service collected 2014 events in total. The users blockchain also had the same integrity results (all 10 of them).

This scenario was tested again, but for one hour. Each service sent over one thousand events during that time period and each blockchain had forty blocks. The integrity was once

Table 5.1: *Logging* & *Auditing* Test 1 (Services)

Services	Number of Events	Number of blocks	Integrity verified?
Sensors	402	14	true
Authorization (RBAC)	400		true
Authorization (ABAC)	401		true
Authentication	404		true
FHIR Gateway	407		true

again verified and all of the results were positive.

To capture the packets exchanged between the local machine and remote machine, a network protocol analyzer was used (WireShark [25]). The HTTP POSTs from the services (that contained the events to be stored) to the *Logging* service had a rate of 0,0017 milliseconds, while the HTTP POSTs from the *Logging* service to the DBMS (storage of events and blocks) had a rate of 0,0003 milliseconds (with the respective responses having a rate of 0,0015 milliseconds) and the HTTP GETs (queries for events and blocks from the *Logging* service to the DBMS) had a rate of 0,0002 milliseconds (with the responses having a rate of 0,0015 milliseconds). The *Logging* service used (in average) 228MB of the remote machine's 4GB RAM and 39% to 46% of the remote machine's 3GHz CPU. The DBMS used 51% to 63% of the machine's CPU and 1.3GB of the RAM.

5.1.4 Customized Logging Test

This test was run for 30 minutes. The results (for the services' blockchain) are available on Table 5.2. Across all services, the average of messages sent to the *Logging* service was 27 per minute (the total amount was 4098).

As with the first test, the users blockchain had the same integrity results (all 10 of them) and the external logging systems validated and stored the correct values for the blockchain data and/or events.

Using WireShark again, the exchanged packets were captured and the HTTP requests (HTTP POSTs) were filtered and those results are represented in table 5.3 (these values are from the time period between the end of the block insertion process and the end of the next block insertion process).

We can see that the DBMS had the most amount of requests since the *Logging* Service

Table 5.2: *Logging* & *Auditing* Test 2 (Services)

Services	Number of Events	Number of blocks	Integrity verified?
Sensors	822	27	true
Authorization (RBAC)	831		true
Authorization (ABAC)	788		true
Authentication	818		true
FHIR Gateway	839		true

Table 5.3: HTTP Requests

Service	Number of Requests
DBMS	575 (68.70%)
Logging	237 (28.32%)
Sensors	2 (0.24%)
Authorization (ABAC)	1 (0.12%)
Authentication	2 (0.24%)
Other *Logging* Service (1)	10 (1.20%)
Other *Logging* Service (2)	10 (1.20%)

not only needs to store values to it but also retrieve them to perform several operations like hash calculation or finding out what was the last checkpoint from the last block inserted. The *Logging* service itself needed to handle the log events sent by the other services, but also the block insertion processes and the other logging services data exchange. Some of the other services had more requests than others, because some of them also needed to validate their log entries before the *Logging* service could insert the respective new block.

A flow graph for the service "FHIR Gateway" log exchanges was also generated (via WireShark), where we filtered every HTTP POST sent to a specific port (8080="*Logging* Service", 9200="ElasticSearch" (DBMS), 8090="Other *Logging* Service (1)", 8091="Other *Logging* Service (2)"). We can see, in Figure 5.4, that whenever the *Logging* Service got a request to store a log into ElasticSearch ("HTTP: POST /logging/postLOG/"), it immediately stored it ("HTTP: POST /logging-servico_5/raw/"). After some time, when the block insertion process started, since the "FHIR Gateway" did not need to verify the logs, the *Logging* service sent the block and respective events to the other *Logging* services (where it got the signed timestamping from the "Other *Logging* Service (2)" first and store that message into ElasticSearch) and after some time and confirming that at least one of the other logging services approved the block, it stored it into the blockchain (Figure 5.5 details the specific packets that represent this flow of events).

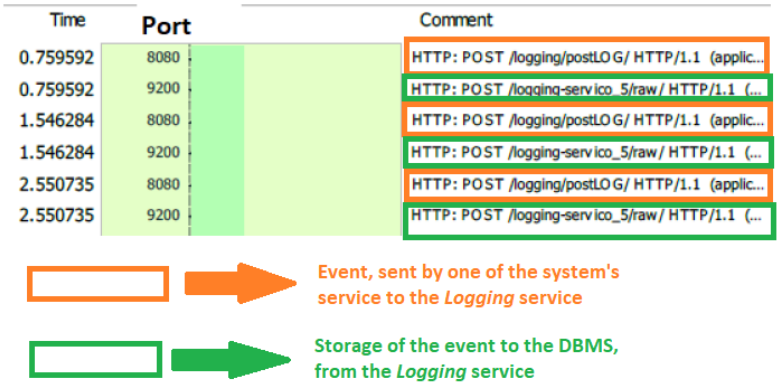


Figure 5.4: Flow graph (log storage).

Figure 5.5: Flow graph (trusted timestamping and block storage).

Other relevant information noted (using Wireshark) was that during the first 9 minutes of this test, the logging service handled almost 4000 HTTP packets (from and to the system services, other logging services and the DBMS) with a rate of 0.0076 milliseconds per packet

Table 5.4: Indexes by number and storage space

Index	**Number of entries**	**Storage (kb)**
FHIR Gateway (Logs)	165	70,5
Authentication (Blockchain)	5	20,73
User B (Blockchain)	6	30,66
FHIR Gateway (Blockchain)	5	20,96
User C (Blockchain)	6	30,73
Authentication (Logs)	159	60,8
Trusted Timestamping (Other *Logging* Service (1))	16	51,2

(the exchanges with the DBMS had a rate of 0.0085 milliseconds) and that the DBMS used (in average) 500MB of the RAM memory (of 8GB total) and 1% to 18% of the CPU (with a speed, at the time, of 3GHz) during the test and the logging process used between 160MB and 195M of the RAM memory and 0% to 4% of the CPU.

Table 5.4 shows, for those 9 minutes, the relation between the number of entries of an index and the storage space that they use, inside Elasticsearch. We can see that, for one service, that has about 165 log events and 5 blocks associated can use up to 90 kilobytes of space (average of 0,43kb for every log entry and 4,2kb for every block). So, if we consider for a single day (24 hours) that we have 5 different services, where each sends, in average, 1000 events and we have 24 blocks (one created for each hour), we will use 2,1MB for the logs and 504kb for the blocks (a total of 2,604MB). In a month (30 days), we could end up with 78,12MB (for 150000 log entries and 3600 blocks) and in a year (365 days) with 950,46MB (for 1825000 log entries and 43800 blocks). Additionally, we could add to that the blockchains that represent the user's timelines on the system and the trusted timestamping messages from the other logging services, which if we follow the 1-hour block creation rule (every 60 minutes a block for one user was generated and stored and one trusted timestamping message also required to be stored from one external logging service), we would have, in a year, (3,2kb for each trusted timestamping entry and and 5,12kb for each user's block, from approximate estimative from table 5.4) approximately 45MB for each user's blockchain and 28MB for each external logging service trusted timestamping of one blockchain. Having 100 users in the system and each of the 5 service's blockchain having trusted timestamping messages stored in the DBMS from 2 external logging services, in a year the total would be (with the 941,7MB mentioned earlier) 5,73GB of storage space needed.

Since we are working with machines with moderate specifications, testing with great amounts of simultaneous messages in short period of times and with many services with different restrictions may lead to some integrity issues (this was referred during section 5.1.2). But, using better specifications, it is possible to have smaller waiting periods and the possibility for handling more messages and services concurrently. With that in mind, these results show that a system will need some extra storage space for every data and metadata, that is needed to prove the integrity of the stream of events that occurred during a time period, and also a moderate RAM and CPU to run everything smoothly. However, the requirements needed are

not very demanding and since the auditing process of these logs can help doctors improve the quality of life of their patients and auditors to detect anomalies in their system without the fear of missing data, the trade-off is worth it.

Using blockchain on a *Logging* service of a platform where the integrity of the events can be life-saving and where there are multiple independent services working at the same time with multiple users, it is possible to prevent tampering of information not only by hackers who want to create discord and ransom certain data but also by system's users who want to hide or cover their activities.

5.2 Auditing with Access Control

5.2.1 Goals

If integrity can be verified, then we can guarantee to the services that sent their logs to us that their data is safe and consistent. Now, we need to prove that our access control truly works and that we do not allow access to the information stored in our DBMS to someone who is not authorized to do so. For that, we will create a policy file with rules that could be applied in real scenarios and try to verify if our Access Control service can manage access requests and give the right answer, according to that file. So, the policy file creation and its integration with the Access Control service will be tested. Test 1 (5.2.3) purpose is to create a policy file and prove the efficiency of log requests by services or FHIR users.

5.2.2 Test Parameters

The subject, resource and environment values that are going to used to create the policy file are the services from sub-section 5.1.2 (from section 5.1), random names (to test the rule adding feature) and resources (patients and respective carers and practitioners) from FHIR. Since we are only interested in the auditing process, the only action available will be reading (the logs, from the environment value). All of the services were deployed in a local machine (*Samsung Laptop, Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 8GB RAM, 1TB Hard Drive, Windows 10 Home 64 bits*). The *Logging* service and its access control are represented in Figure 5.2 and the services/auditors communicate with them in the same way as the other services from that diagram (including the DBMS communication).

How the messages are exchanged between the different entities involved in the auditing process are shown in Figure 5.6. A service will send a request to the *Logging* service, so it can receive the log events from several users that a certain subject has permission to access. For every user of the request, the *Logging* service will send a message to the PEP that distinguishes the different attributes and respective values (`subject`, `resource` (one of the users), `environment` (the service where the respective logs are stored) and `action` ("read")). The PEP adapts the message to the XACML language and forwards the message to the PDP that will analyze it and compare its values to the policy file to reach a decision (*Permit* or *Deny*). It sends the response to the PEP, that will send the response to the *Logging* service.

After sending every request, for each user, the *Logging* service will gather all of the authorized log events (from the DBMS) and send it to the service that made the request.

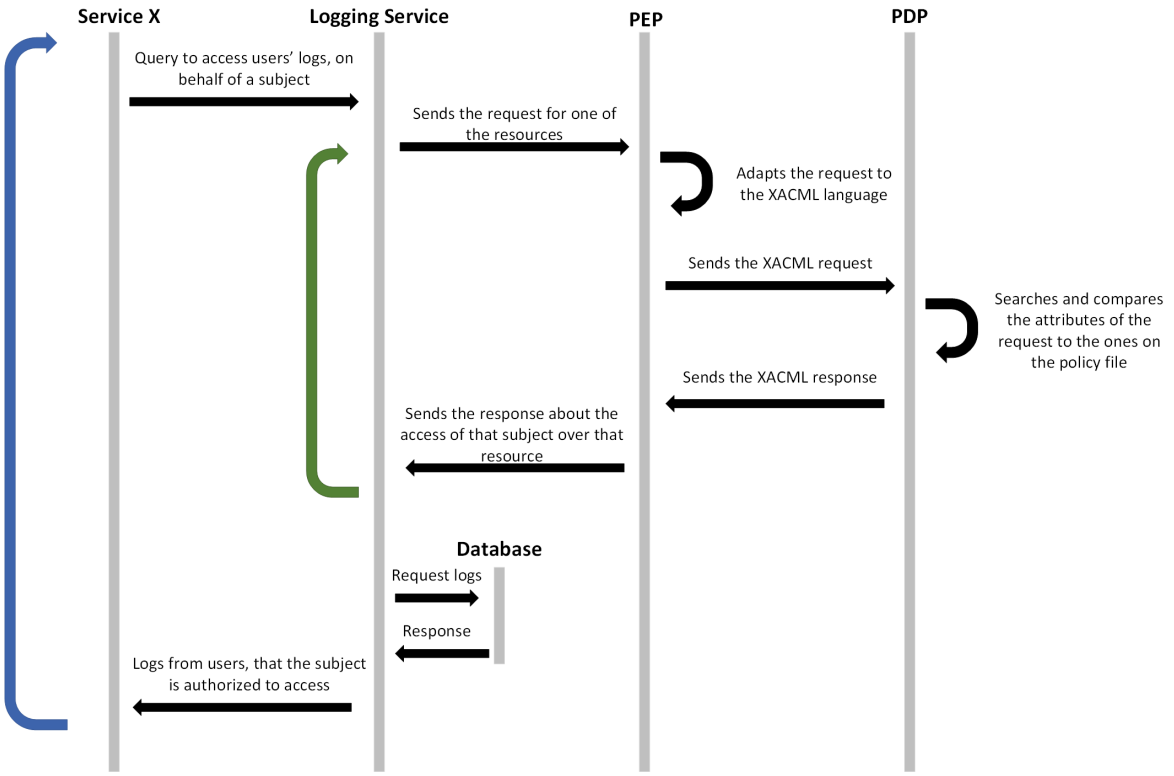


Figure 5.6: Flux of events for section 5.2

5.2.3 Auditing from Services and FHIR Resources

To create the policy file, a program was developed that would create rules for every of the five services (the ones referred on the second test of section 5.1) and for fifteen different *Practitioners* (where the relevant data was retrieved from the FHIR resources).

Figure 5.7 shows the logic behind the policy file rule creation process for a practitioner as the subject (for a service, instead of using the role, the rules must be defined by the other services, for example, service "A" allows service "B" and "C" to access its logs). We also wanted to represent the scenario where some services or other authorized parties could have the possibility to deny certain accesses of certain subjects to resources, even if their role, generically, would allow them that access (in the figure, in the first rule, even though the Practitioner is the "Medical Expert" of the Patients "A", "B", "C" and "D", he cannot access the service "Y" logs of the Patients "A" and "B", since that was a restriction imposed by the facility where the Practitioner works at).

Every rule has one `subject` (person or service) and several sub-rules, one for each `environment` (service) and corresponding `resources` (users of the system). For example, analyzing the policy file, the third rule, the `subject` "Sasikala S" (a practitioner with several

Table 5.5: Policy's Rule #3

Sub-Rule	Number of Resources	Environment
1	5	Authorization (ABAC)
2	4	Sensors
3	8	FHIR Gateway

Table 5.6: Policy's Rule #9

Sub-Rules	Number of Resources	Environment
1	7	Sensors
2	9	Authentication
3	3	Authorization (ABAC)
4	6	Authorization (RBAC)

different roles over different patients, whose FHIR ID is 221712 *) can access some logs of her Patients, that are depicted on Table 5.5 and for the ninth rule, with the `subject` "Authentication", we have the results shown on Table 5.6.

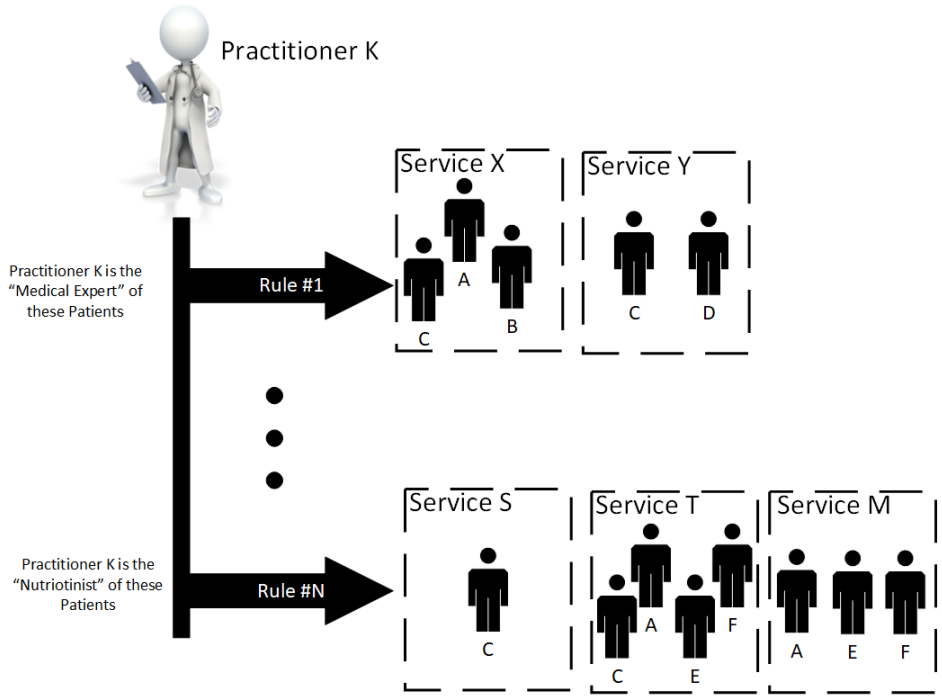


Figure 5.7: Policy construction method

In table 5.6, it can be verified that it was created 4 sub-rules, each one with a specific set of resources and action and environment associated. With this structure in mind, we started sending requests to our PEP with different values (Table 5.7).

Every decision on Table 5.7 was the one expected according to the policy file. Because

*"Sasikala S" HAPI FHIR resource url: `http://hapi.fhir.org/baseDstu3/Practitioner?_id=221712` (visited on 07/07/2018)

Table 5.7: ABAC Test 1 (Rule #9)

Test	Subject	Resource	Action/Environment	Decision
1	Authentication	User A	"read"/"Authorization (ABAC)"	Permit
2	Authentication	User A	"read"/"Sensors"	Deny
3	Authentication	User B	"read"/"Sensors"	Permit
4	Authentication	User B	"read"/"Authorization (ABAC)"	Permit
5	Sensors	User B	"read"/"Authorization (ABAC)"	Deny

of the clear distinction between attributes, even if a certain `subject` has a role over two `resources`, it does not mean that he can access the same `environment` information where those two `resources` have data stored, and we can see that separation on the table's Test 1 and Test 2 (the `subject` can access "User A" logs on "Authorization (ABAC)" but not on "Sensors") or Test 2 and Test 3 (the `subject` can access "User B" logs on "Sensors" but cannot access "User A" logs on the same `environment`). Also, not every `subject` can do the same actions over the same `resource` and `environment` as other `subjects` (Test 4 and Test 5, `subject` "Authentication" can access "User B" logs on "Authorization (ABAC)", but `subject` "Sensors" cannot). Analyzing the packets exchanged (using Wireshark), it was verified that (in average) the HTTP packets were transmitted with a rate of 0,0031 milliseconds (request and response included), so if the *Logging* service would send one message to the PEP and it would forward it to the PDP and then each one would send the respective response until reaching the *Logging* service, that would take about 0,0062 milliseconds. That means that the *Logging* service would know if it could allow access to a resource for a subject in 6,2μs. Also, 150MB of RAM (of 8GB total) and 0% to 0,2% of CPU (speed of 3GHz) were used during this auditing testing phase for the *Logging* service and ABAC (PEP and PDP).

We can conclude that the ABAC, using the XACML framework, is a reliable tool to handle restrictive accesses to data from a service or system, since it uses several distinct attributes to identify and differentiate the participants (`subject` and `resource`) of the authorization query, their intentions and other possible factors, before reaching a decision. It also does it in a quick manner, unnoticeable to the human perception.

Investing on the development of an access control integrated with the FHIR standard should be a very important concern by many, since it handles healthcare information and the exchange of that type of data should be meticulous controlled.

Using XACML proves to be effective, since it can differentiate several attributes and we can be precise on knowing who should access what from whom according to certain conditions. With our tests we were able to create rules for each subject and what type of service it could access from specific resources. Also, it can handle many requests at the same time, it can reach a decision and send an answer on a very short amount of time (which is useful when we get one request to access several resources and we can communicate quickly with the PDP by sending one message per every resource of the request) and has the possibility of working with almost every service using a different kind of programming language (interoperability).

CHAPTER 6

Conclusion

This chapter concludes the document and points out some aspects of the proposed solution that can be improved upon and its implementation.

Through this document we discussed how the logging and auditing is usually developed into a system and the problems that come with it if we do not adapt them to current standards, since they might be vulnerable to attackers. We also highlighted several security frameworks that are used for specific purposes, that normally are not associated with either logging and auditing services. By adapting these different standards and technologies into these services we can eliminate some problems associated with the standard logging and auditing services, including integrity and log access issues. So, by adapting the blockchain technology for logging we can guarantee the integrity of the stream of log events and detect tampering attempts by malicious users through the comparison of hash values and by integrating an attribute-based access control, using the XACML standard, we can allow a subject's access to certain logs from specific users and services (using attributes that can make all of those distinctions), for the auditing process. Also, this solution can be adapted into different type of services and not only for logging and auditing, due to the flexibility of the blockchain technology and XACML standard.

After the analysis of the Chapter 5 results, it can be concluded that there is some basis for a fully operational and efficient logging service that uses an emerging technology to maintain the integrity of these events. Adding to that a robust access control that can integrate a very specific and detailed set of rules that will allow for a more secure auditing process, we can have one of the most secure platforms out there and make practically impossible for someone to tamper with the information that we have store on the system.

6.1 Future Work

Still, more work needs to be done and several tests must be fulfilled before actually applying this implementation into a platform. Some of the things that need to be thought

about is the scalability of the events (How many events are we going to have in a single day? Will we need a better way to store all of that information?), the block creation and insertion trigger (Should it be time relative or by quantity of events), access control aspects (How much complex should this be? Should we implement very detailed restrictions or more generic ones?) and the efficiency and need of a blockchain into a service (Is it really necessary to have a blockchain technology for a logging service? Is it worth the performance cost, due to the several operations that are involved with the blockchain technology, like hashing and the extra storage space for the block's metadata?).

All of this needs to be pondered but we should not disregard the importance of the logging service in maintaining a system secure. Smart hackers know this and will attack it to assure that they will never be found out. So, investing on a more complex logging may be worth it even if that means some extra work on the part of the developers and some higher costs.

References

[1] M. A. B. Ahmadon, S. Yamaguchi, S. Saon, and A. K. Mahamad, "On service security analysis for event log of IoT system based on data petri net", in *2017 IEEE International Symposium on Consumer Electronics (ISCE)*, IEEE, Nov. 2017. DOI: 10.1109/isce.2017.8355531.

[2] Alentum Software Ltd., *WebLog Expert*. [Online]. Available: https://www.weblogexpert.com/ (visited on 07/07/2018).

[3] R. A. Alimbuyog, J. C. D. Cruz, and R. V. Sevilla, "Development of motorcycle data logging system with visual basic data simulation for accident analysis", in *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, IEEE, Dec. 2017. DOI: 10.1109/hnicem.2017.8269482.

[4] L. Aniello, R. Baldoni, E. Gaetani, F. Lombardi, A. Margheri, and V. Sassone, "A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database", in *2017 13th European Dependable Computing Conference (EDCC)*, IEEE, Sep. 2017. DOI: 10.1109/edcc.2017.31.

[5] Apache Software Foundation, *Apache Tomcat*. [Online]. Available: http://tomcat.apache.org/ (visited on 07/07/2018).

[6] B. Armstrong and F. Ehrsam, *COINBASE DIGITAL API*. [Online]. Available: https://developers.coinbase.com/ (visited on 07/07/2018).

[7] M. Armstrong, *The Price Tag Attached to Data Breaches*, statista, Jun. 2017. [Online]. Available: https://www.statista.com/chart/9918/the-price-tag-attached-to-data-breaches/ (visited on 07/07/2018).

[8] A. M. Atiq and L. A. Alsulaiman, "Using XACML to enhance compliance with privacy regulations in health sector", in *2016 World Symposium on Computer Applications & Research (WSCAR)*, IEEE, Mar. 2016. DOI: 10.1109/wscar.2016.25.

[9] Australia, *National E-Health Transition Authority*, Jul. 2005. [Online]. Available: https://www.digitalhealth.gov.au/ (visited on 07/07/2018).

[10] Auth0, *JSON Web Tokens*. [Online]. Available: https://jwt.io/ (visited on 07/07/2018).

[11] S. Banon, *ElasticSearch*. [Online]. Available: https://www.elastic.co/ (visited on 07/07/2018).

[12] M. Baum, R. Das, and E. Swan, *Splunk*. [Online]. Available: https://www.splunk.com/ (visited on 07/07/2018).

[13] *Bitcoin*, 2009. [Online]. Available: https://www.bitcoin.com/ (visited on 07/07/2018).

[14] BitPay, Inc., *Bitcore: Bitcoin Platform and API*. [Online]. Available: https://bitcore.io/ (visited on 07/07/2018).

[15] L. Bohn, *Organization for the Advancement of Structured Information Standards*. [Online]. Available: https://www.oasis-open.org/ (visited on 07/07/2018).

[16] Brazil, *Vitoria City Hall*. [Online]. Available: http://www.vitoria.es.gov.br/ (visited on 07/07/2018).

[17] Breach Level Index, *Data Breach Reports and Other Resources*. [Online]. Available: https://breachlevelindex.com/data-breach-library (visited on 07/07/2018).

[18] G. Brewer, *OpenID Usage Statistics*, BuiltWith. [Online]. Available: https://trends.builtwith.com/docinfo/OpenID (visited on 07/07/2018).

[19] B. Campbell, J. Bradley, N. Sakimura, and T. Lodderstedt, *OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens*, Jan. 2018. [Online]. Available: https://tools.ietf.org/id/draft-ietf-oauth-mtls-07.html (visited on 07/07/2018).

[20] Q. Cao, Y. Qiao, and Z. Lyu, "Machine learning to detect anomalies in web log analysis", in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, IEEE, Dec. 2017. DOI: 10.1109/compcomm.2017.8322600.

[21] *Cardano*, 2017. [Online]. Available: https://www.cardano.org/ (visited on 07/07/2018).

[22] R. Chen, W. Ji, S. Duan, Q. Ling, and F. Li, "A novel method to analyze logs generated by wireless telecommunication systems", in *2017 2nd International Conference on Advanced Robotics and Mechatronics (ICARM)*, IEEE, Aug. 2017. DOI: 10.1109/icarm.2017.8273166.

[23] coindesk, *Bitcoin (USD) Price*. [Online]. Available: https://www.coindesk.com/price/ (visited on 07/07/2018).

[24] Colored Coins, *ColoredCoins.org official repository*. [Online]. Available: https://github.com/Colored-Coins (visited on 07/07/2018).

[25] G. Combs, *WireShark*. [Online]. Available: https://www.wireshark.org/ (visited on 07/07/2018).

[26] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging", in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09, Berkeley, CA, USA: USENIX Association, 2009, pp. 317–334. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855768.1855788.

[27] M. Dahmane and S. Foucher, "Combating insider threats by user profiling from activity logging data", in *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, IEEE, Apr. 2018. DOI: 10.1109/icdis.2018.00039.

[28] C. Dangerville, *AuthZForce*. [Online]. Available: https://github.com/authzforce (visited on 07/07/2018).

[29] ——, *AuthzForce PDP features page*. [Online]. Available: http://authzforce-ce-fiware.readthedocs.io/en/latest/Features.html (visited on 07/07/2018).

[30] ——, *AuthzForce RESTful PDP*. [Online]. Available: https://github.com/authzforce/restful-pdp (visited on 07/07/2018).

[31] M. S. M. Deli, S. A. Ismail, N. Kama, O. M. Yusop, A. Azmi, and Y. Yahya, "Malware log files for internet investigation using hadoop: A review", in *2017 IEEE Conference on Big Data and Analytics (ICBDA)*, IEEE, Nov. 2017. DOI: 10.1109/icbdaa.2017.8284112.

[32] Elasticsearch, *Mapping*. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html (visited on 07/07/2018).

[33] Elasticsearch BV, *Kibana*. [Online]. Available: https://www.elastic.co/products/kibana (visited on 07/07/2018).

[34] ——, *Logstash*. [Online]. Available: https://www.elastic.co/products/logstash (visited on 07/07/2018).

[35] ——, *X-Pack*. [Online]. Available: https://www.elastic.co/products/x-pack (visited on 07/07/2018).

[36] England, *NHS Digital*, Apr. 2013. [Online]. Available: https://digital.nhs.uk/ (visited on 07/07/2018).

[37] *Ethereum*, 2015. [Online]. Available: https://ethereum.org/ (visited on 07/07/2018).

[38] Facebook, *Facebook Login for the Web with the JavaScript SDK*. [Online]. Available: https://developers.facebook.com/docs/facebook-login/web/ (visited on 07/07/2018).

[39] A. Gandhi, D. Frey, P. Sundar, J. Heyman, and M. McCambridge, "Design of a cellular-enabled data-logging system for wheelchair use characterization", in *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, IEEE, Oct. 2017. DOI: 10.1109/ghtc.2017.8239316.

[40] Y. Gao, Y. Ma, and D. Li, "Anomaly detection of malicious users' behaviors for web applications based on web logs", in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, IEEE, Oct. 2017. DOI: 10.1109/icct.2017.8359854.

[41] Github, *Using GitHub OAuth*. [Online]. Available: https://help.github.com/enterprise/2.13/admin/guides/user-management/using-github-oauth/ (visited on 07/07/2018).

[42] V. Goel and N. Perlroth, *Yahoo Says 1 Billion User Accounts Were Hacked*, The New Work Times, Dec. 2016. [Online]. Available: https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html (visited on 07/07/2018).

[43] Google, *Using OAuth 2.0 to Access Google APIs*. [Online]. Available: https://developers.google.com/identity/protocols/OAuth2 (visited on 07/07/2018).

[44] A. Greenberg, *A Texas Jury's Guilty Verdict Should Worry IT Admins*, Wired, Jun. 2016. [Online]. Available: https://www.wired.com/2016/06/texas-jurys-guilty-verdict-worry-admins/ (visited on 07/07/2018).

[45] E. Hammer and E. Lahav, *The OAuth 1.0 Protocol*, Apr. 2010. [Online]. Available: https://tools.ietf.org/html/rfc5849 (visited on 07/07/2018).

[46] G. Hartung, "Secure audit logs with verifiable excerpts", in *Topics in Cryptology - CT-RSA 2016*, Springer International Publishing, 2016, pp. 183–199. DOI: 10.1007/978-3-319-29485-8_11.

[47] A. Hasiloglu and A. Bali, "Central audit logging mechanism in personal data web services", in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, IEEE, Mar. 2018. DOI: 10.1109/isdfs.2018.8355333.

[48] HL7, *Fast Healthcare Interoperability Resources*. [Online]. Available: https://www.hl7.org/fhir/ (visited on 07/07/2018).

[49] ——, *FHIR Resources*. [Online]. Available: https://www.hl7.org/fhir/resourcelist.html (visited on 07/07/2018).

[50] ——, *FHIR Security*. [Online]. Available: https://www.hl7.org/fhir/security.html (visited on 07/07/2018).

[51] ——, *FHIR Security Labels*. [Online]. Available: https://www.hl7.org/fhir/security-labels.html (visited on 07/07/2018).

[52] ——, *FHIR Summary*. [Online]. Available: http://www.hl7.org/fhir/DSTU1/fhir-summary.pdf (visited on 07/07/2018).

[53] ——, *HAPI FHIR*. [Online]. Available: http://fhirtest.uhn.ca/ (visited on 07/07/2018).

[54] ——, *Organizations interested in FHIR*. [Online]. Available: http://wiki.hl7.org/index.php?title=Organizations_interested_in_FHIR (visited on 07/07/2018).

[55] ——, *Patient 3381806*. [Online]. Available: http://hapi.fhir.org/baseDstu3/Patient/3381806 (visited on 07/07/2018).

[56] ——, *Use Cases*. [Online]. Available: https://www.hl7.org/fhir/usecases.html (visited on 07/07/2018).

[57] R. A. Hoffman, H. Wu, J. Venugopalan, P. Braun, and M. D. Wang, "Intelligent mortality reporting with FHIR", *IEEE Journal of Biomedical and Health Informatics*, pp. 1–1, 2018. DOI: 10.1109/jbhi.2017.2780891.

[58] V. Holub and T. Parsons, *Logentries*. [Online]. Available: https://logentries.com/ (visited on 07/07/2018).

[59] N. Hong, K. Wang, L. Yao, and G. Jiang, "Visual FHIR: An interactive browser to navigate HL7 FHIR specification", in *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, IEEE, Aug. 2017. DOI: 10.1109/ichi.2017.54.

[60] D. Huemer and A. M. Tjoa, "A stepwise approach towards an interoperable and flexible logging principle for audit trails", in *2010 Seventh International Conference on Information Technology: New Generations*, IEEE, 2010. DOI: 10.1109/itng.2010.33.

[61] J. H. Huh and A. Martin, "Trusted logging for grid computing", in *2008 Third Asia-Pacific Trusted Infrastructure Technologies Conference*, IEEE, Oct. 2008. DOI: 10.1109/aptc.2008.9.

[62] IANA, *JSON Web Token (JWT)*. [Online]. Available: https://www.iana.org/assignments/jwt/jwt.xhtml (visited on 07/07/2018).

[63] O. M. Ilhan, D. Thatmann, and A. Kupper, "A performance analysis of the XACML decision process and the impact of caching", in *2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, IEEE, Nov. 2015. DOI: 10.1109/sitis.2015.83.

[64] InfoBeyond Technology, *Security Policy Tool*. [Online]. Available: https://securitypolicytool.com/product (visited on 07/07/2018).

[65] Z. R. Janki, Z. Szabo, V. Bilicki, and M. Fidrich, "Authorization solution for full stack FHIR HAPI access", in *2017 IEEE 30th Neumann Colloquium (NC)*, IEEE, Nov. 2017. DOI: 10.1109/nc.2017.8263266.

[66] D. E. Kateb, T. Mouelhi, Y. L. Traon, J. Hwang, and T. Xie, "Refactoring access control policies for performance improvement", in *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12*, ACM Press, 2012. DOI: 10.1145/2188286.2188346.

[67] Kenya, *PCEA Kikuyu Hospital*, 1908. [Online]. Available: http://pceakikuyuhospital.org/ (visited on 07/07/2018).

[68] S.-H. Kim, D.-H. Kim, and D.-S. Kim, "Event log analysis software design for naval combat system using smart platform", in *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, IEEE, Jan. 2018. DOI: 10.23919/elinfocom.2018.8330563.

[69] G. Magyar, "Blockchain: Solving the privacy and research availability tradeoff for EHR data: A new disruptive technology in health data management", in *2017 IEEE 30th Neumann Colloquium (NC)*, IEEE, Nov. 2017. DOI: 10.1109/nc.2017.8263269.

[70] I. McCrae, *Orion Health*, 1993. [Online]. Available: https://digital.nhs.uk/ (visited on 07/07/2018).

[71] S. de Medeiros Camara, L. F. R. da Costa Carmo, and L. Pirmez, "Towards a storage-efficient and categorized secure log structure scheme for embedded systems", in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, IEEE, Nov. 2017. DOI: 10.1109/dasc-picom-datacom-cyberscitec.2017.96.

[72] W. Meng, E. Tischhauser, Q. Wang, Y. Wang, and J. Han, "When intrusion detection meets blockchain technology: A review", *IEEE Access*, pp. 1–1, 2018. DOI: 10.1109/access.2018.2799854.

[73] Microsoft, *Authorize access to Azure Active Directory web applications using the OAuth 2.0 code grant flow*. [Online]. Available: https://docs.microsoft.com/pt-pt/azure/active-directory/develop/active-directory-protocols-oauth-code (visited on 07/07/2018).

[74] Mkyong, *Spring Boot – Deploy WAR file to Tomcat*. [Online]. Available: https://www.mkyong.com/spring-boot/spring-boot-deploy-war-file-to-tomcat/ (visited on 07/07/2018).

[75] MSConsulting, *Federated SSO, A Primer (SAML, OAuth 2.0, OpenID Connect)*. [Online]. Available: https://www.mandsconsulting.com/federated-sso-a-primer-saml-oauth-2-0-openid-connect/ (visited on 07/07/2018).

[76] S. Nair, *Axiomatics, XACML Reference Architecture*, Nov. 2013. [Online]. Available: https://www.axiomatics.com/blog/xacml-reference-architecture/ (visited on 07/07/2018).

[77] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. Princeton University Press, 2016, ISBN: 978-0691171692.

[78] NextLabs, *NextLabs Integrates Data Classification, Access Control, Rights Management and Data Loss Prevention using the XACML Standard*, Feb. 2012. [Online]. Available: https://www.nextlabs.com/press/nextlabs-integrates-data-classification-access-control-rights-management-and-data-loss-prevention-u/ (visited on 07/07/2018).

[79] F. Ning, Y. Wen, G. Shi, and D. Meng, "Efficient tamper-evident logging of distributed systems via concurrent authenticated tree", in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, IEEE, Dec. 2017. DOI: 10.1109/pccc.2017.8280476.

[80] OASIS, *A Brief Introduction to XACML*. [Online]. Available: https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html (visited on 07/07/2018).

[81] ——, *Security Assertion Markup Language*. [Online]. Available: https://wiki.oasis-open.org/security/FrontPage (visited on 07/07/2018).

[82] ——, *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. [Online]. Available: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html (visited on 07/07/2018).

[83] ——, *XACML*, Jan. 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html (visited on 07/07/2018).

[84] T. Ong, S. Mukherjee, and H. Shirazi, *Attribute-Based Access Control (ABAC) on FHIR*. [Online]. Available: https://devpost.com/software/attribute-based-access-control-abac-on-fhir (visited on 07/07/2018).

[85] Ontario, *Ministry of Health and Long-Term Care*, 1882. [Online]. Available: http://www.health.gov.on.ca/ (visited on 07/07/2018).

[86] OpenStack Foundation, *OpenStack*, Oct. 2010. [Online]. Available: https://www.openstack.org/ (visited on 07/07/2018).

[87] Oracle, *Package java.util.logging*. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/logging/package-summary.html (visited on 07/07/2018).

[88] A. Parecki and C. Messina, *OAuth 2.0*. [Online]. Available: https://oauth.net/2/ (visited on 07/07/2018).

[89] ——, *OAuth Community Site*. [Online]. Available: https://oauth.net/ (visited on 07/07/2018).

[90] C. Petersen and P. Villella, *LogRhythm*. [Online]. Available: https://logrhythm.com/ (visited on 07/07/2018).

[91] Pivotal Software, *Spring Boot*. [Online]. Available: http://spring.io/projects/spring-boot (visited on 07/07/2018).

[92] D. Pol, *OAUTHORIZATION*, Jul. 2015. [Online]. Available: https://deepakpol.wordpress.com/2015/07/16/oauthorization/ (visited on 07/07/2018).

[93] Portugal 2020, *COMPETE 2020*. [Online]. Available: http://www.poci-compete2020.pt/ (visited on 07/07/2018).

[94] Python Software Foundation, *Logging HOWTO*. [Online]. Available: https://docs.python.org/2/howto/logging.html (visited on 07/07/2018).

[95] D. Rahmawati, R. Sarno, C. Fatichah, and D. Sunaryono, "Fraud detection on event log of bank financial credit business process using hidden markov model algorithm", in *2017 3rd International Conference on Science in Information Technology (ICSITech)*, IEEE, Oct. 2017. DOI: 10.1109/icsitech.2017.8257082.

[96] Relias, *13 hospital workers fired for snooping in Britney Spears' medical records*, May 2008. [Online]. Available: https://www.ahcmedia.com/articles/11576-13-hospital-workers-fired-for-snooping-in-britney-spears-medical-records (visited on 07/07/2018).

[97] Renal and Urology News, *Staff Nurse Faces Jail Time for HIPAA Violations*. [Online]. Available: http://www.renalandurologynews.com/legal-issues-in-medicine/nurse-jail-hipaa-violations/article/119854/2/ (visited on 07/07/2018).

[98] *Ripple*, 2012. [Online]. Available: https://ripple.com/ (visited on 07/07/2018).

[99] M. Rosbach, *Privacy breach at Yakima Virginia Mason Memorial hospital affects 419 patients*, The Seattle Times, Apr. 2017. [Online]. Available: https://www.seattletimes.com/seattle-news/privacy-breach-at-yakima-virginia-mason-memorial-hospital-affects-419-patients/ (visited on 07/07/2018).

[100] J. Ruminski, A. Bujnowski, T. Kocejko, A. Andrushevich, M. Biallas, and R. Kistler, "The data exchange between smart glasses and healthcare information systems using the HL7 FHIR standard", in *2016 9th International Conference on Human System Interactions (HSI)*, IEEE, Jul. 2016. DOI: 10.1109/hsi.2016.7529684.

[101] N. Sakimura, J. Bradley, M. Jones, and G. Fletcher, *OpenID Connect*. [Online]. Available: http://openid.net/connect/ (visited on 07/07/2018).

[102] ——, *OpenID Foundation website*. [Online]. Available: https://openid.net/ (visited on 07/07/2018).

[103] R. Samavi and M. P. Consens, "Publishing privacy logs to facilitate transparency and accountability", *Journal of Web Semantics*, Feb. 2018. DOI: 10.1016/j.websem.2018.02.001.

[104] Y. K. R. Sanchez, S. A. Demurjian, and M. S. Baihan, "Achieving RBAC on RESTful APIs for mobile apps using FHIR", in *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, IEEE, Apr. 2017. DOI: 10.1109/mobilecloud.2017.22.

[105] S. Saraswat, *Former UCLA Medical Center employee Huping Zhou sentenced jail time for looking at private medical files*, Daily Bruin, May 2010. [Online]. Available: http://dailybruin.com/2010/05/05/former-ucla-medical-center-employee-huping-zhou-se/ (visited on 07/07/2018).

[106] Saudi Arabia, *Ministry of Health*, 1950. [Online]. Available: https://www.moh.gov.sa/en/Pages/default.aspx (visited on 07/07/2018).

[107] A. Sawabe, H. Yoshida, and K. Nogami, "Log analysis in a HTTP proxy server for accurately estimating web QoE", in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2018. DOI: 10.1109/ccnc.2018.8319210.

[108] S. Scott, *These 15 Tech Companies Chose the ELK Stack Over Proprietary Logging Software*. [Online]. Available: https://logz.io/blog/15-tech-companies-chose-elk-stack/ (visited on 07/07/2018).

[109] D. W. Simborg, *Health Level 7*, 1987. [Online]. Available: http://www.hl7.org/ (visited on 07/07/2018).

[110] S. P. Singh and Meenu, "Analysis of web site using web log expert tool based on web data mining", in *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, IEEE, Mar. 2017. DOI: 10.1109/iciiecs.2017.8275961.

[111] SmartBear Software, *Swagger*. [Online]. Available: https://swagger.io/ (visited on 07/07/2018).

[112] P. Smith and N. Cary, *Blockchain API: Bitcoin API*. [Online]. Available: https://www.blockchain.com/api (visited on 07/07/2018).

[113] T. Sridhar, V. Vivek, and R. Shekhar, "Seclogmon : Security in cloud computing using activity log for consumer data protection", in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, IEEE, May 2017. DOI: 10.1109/rteict.2017.8256839.

[114] Stackify, *Retrace*. [Online]. Available: https://stackify.com/retrace/ (visited on 07/07/2018).

[115] M. I. Sukmana, K. A. Torkura, F. Cheng, C. Meinel, and H. Graupner, "Unified logging system for monitoring multiple cloud storage providers in cloud storage broker", in *2018 International Conference on Information Networking (ICOIN)*, IEEE, Jan. 2018. DOI: 10.1109/icoin.2018.8343081.

[116] S. Suzuki and J. Murai, "Blockchain as an audit-able communication channel", in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, IEEE, Jul. 2017. DOI: 10.1109/compsac.2017.72.

[117] C. Thompson, *US government charges 3 people for the biggest bank hack in history*, Business Insider, Nov. 2015. [Online]. Available: http://www.businessinsider.com/jpmorgan-hack-2015-11 (visited on 07/07/2018).

[118] Twitter, *Oauth with the Twitter APIs*. [Online]. Available: https://developer.twitter.com/en/docs/basics/authentication/overview/oauth (visited on 07/07/2018).

[119] Universidade de Aveiro OAuth, *Protocolo Open Authorization*. [Online]. Available: https://identity.ua.pt/oauth/ (visited on 07/07/2018).

[120] S. Varadhan, *India bank hack similar to (dollar)81 million Bangladesh central bank heist*, Reuters, Feb. 2018. [Online]. Available: https://www.reuters.com/article/us-city-union-bank-swift/india-bank-hack-similar-to-81-million-bangladesh-central-bank-heist-idUSKCN1G319K (visited on 07/07/2018).

[121] ViewDS, *Access Sentinel Authorization Server*. [Online]. Available: http://www.viewds.com/products/access-sentinel.html (visited on 07/07/2018).

[122] P. Vimalachandran, H. Wang, Y. Zhang, B. Heyward, and F. Whittaker, "Ensuring data integrity in electronic health records: A quality health care implication", in *2016 International Conference on Orange Technologies (ICOT)*, IEEE, Dec. 2016. DOI: 10.1109/icot.2016.8278970.

[123] C. Wickramage, C. Fidge, T. Sahama, and R. Wong, "Challenges for log based detection of privacy violations during healthcare emergencies", in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, IEEE, Dec. 2017. DOI: 10.1109/glocom.2017.8254433.

[124] Wikipedia, *Trusted timestamping*. [Online]. Available: https://en.wikipedia.org/wiki/Trusted_timestamping (visited on 07/07/2018).

[125] S. Yang, F. Hadiji, K. Kersting, S. Grannis, and S. Natarajan, "Modeling heart procedures from EHRs: An application of exponential families", in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, IEEE, Nov. 2017. DOI: 10.1109/bibm.2017.8217696.

[126] W. Zhang, D. Vecchio, G. Wasson, and M. Humphrey, "Flexible and secure logging of grid data access", in *2006 7th IEEE/ACM International Conference on Grid Computing*, IEEE, 2006. DOI: 10.1109/icgrid.2006.311001.

[127] Y. Zhang and B. Zhang, "A new testing method for XACML 3.0 policy based on ABAC and data flow", in *2017 13th IEEE International Conference on Control & Automation (ICCA)*, IEEE, Jul. 2017. DOI: 10.1109/icca.2017.8003052.

Appendix

Real-life Security Failures

On this topic, we will present some cases where the a system's security was breached and how much that cost the organization and people involved so that it can be demonstrated how important a robust and secure implementation must be. It will also be discussed some hypothetical situations that could happen if there is not much care on securing a system.

There are several examples of real life situations where a lack of effort in developing a robust system caused severe problems. Yahoo, in 2016, revealed that over 500 million of its users private information had been exposed. However, later on, it was found out that similar cases happened since 2013 and almost at the end of 2017 those numbers were up to 3 billion user accounts. Because of all of these incidents, Yahoo's sale price was reduced to less than three quarters (3/4) of its original price [42]. In 2010, surgeon Huping Zhou was caught accessing (illegally) the University of California (Los Angeles) medical records system over 300 times. He was fired, served four months in jail and paid a $2,000 fine [105]. One nurse once checked a patient's file and gave that private information to the husband (of the patient). Later, when that patient found out, that nurse was accused of illegal sharing of information and was faced with a $250,000 fine and up to 10 years in prison [97]. During Britney Spears' infamous controversies in 2008, several people (including non-medical staff) managed to access her private medical records. The criminal fines discussed for that privacy invasion were up to $250,000 [96]. Several workers from Virginia Mason Memorial Hospital accessed unauthorized data from over 400 patients and got access to many personal and private information (the privacy officer at the time guaranteed that the most private information accessed was the Social Security numbers). Even though some of them were immediately fired, the hospital's reputation was damaged [99]. The Bangladesh's central bank lost more than $80 million to hackers, who cleverly hidden their tracks by sending malware to their printers [120]. JPMorgan Chase & Co. was once compromised, because hackers were able to remove and alter certain records from the banks' computer systems [117]. ClickMotive, a auto dealership software firm, back in 2011 had several of its files deleted because of an systems administrator. Since they had all of that data in one place, it was lost forever [44].

With these cases we can see that security breaches can result on monetary losses that go from thousands to millions of dollars, companies to go bankrupt or closing down, private

information that goes public or lost forever and people losing their jobs and/or getting arrested and earning a negative reputation (that can result on not getting hired again).

Unfortunately, there could also exist scenarios where it is possible to completely hide illegal activities and certain crimes could go unpunished. Someone could manage to access the system log's location and manipulate it, to hide their tracks and that could be possible if they edited or deleted certain events. Another scenario is when one service could manage to post or retrieve (log) information of another service, inside the same system. All of this leads to privacy and security breaches.

Making sure that any kind of project (either that be an application, a website or something else) can manage to defend itself from multiple security breaches attempts is fundamental, not only to show current and potential clients that the system's data is safe, but to also avoid potential lawsuits that could lead to the shutdown of that project and the life-affecting consequences of everyone involved.

Knowing that (according to statista [7]) data breaches can have costs over one million dollars to a company (averages of, in millions, \$7.4 in the USA, \$4.3 in Canada and \$3.7 in Germany, during the year 2017) it can be concluded that there is a problem in data security and knowing that in 2017, there were over two billions of data records compromised and 27% of the breach incidents happened on the healthcare industry (source: Breach Level Index [17]) and that over the years more and more information is accessible online, developing new ways to fight against these transgressions becomes more vital than ever.